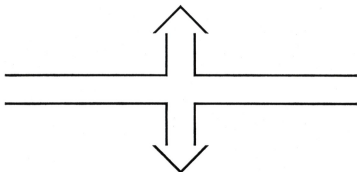
森亮一 監修
マイクロコンピュータシリーズ 14

68000マイクロコンピュータ

喜田祐三
萩原吉宗
岩崎一彦
著

丸善株式会社

森亮一 監修
マイクロコンピュータシリーズ **14**

68000マイクロコンピュータ

喜田祐三
萩原吉宗
岩崎一彦
著

丸善株式会社

序

1971年マイクロコンピュータが誕生してから，すでに10年が過ぎた．そして，今日，マイクロコンピュータは広範な分野において多様な形で私達の社会生活を支えるようになったといっても過言ではない．今や，私達の生活の中には無意識のうちにマイクロコンピュータの恩恵を受けている部分がいかに多いか知るべきである．さて，マイクロコンピュータはこの10年間，MOS-LSI技術の進歩に支えられて，集積度，スピードはともに30〜50倍改善され，また消費電力も1/10以下に低減化された．前者は主としてLSIの微細化技術の進歩に負うところが大きく，一方後者はデバイス，回路技術の進歩に負うところが大きい．マイクロコンピュータは大別すると (1) 多様能シングルチップ，(2) 高性能マルチチップ，の2つに分類される．前者は主として家電・民生分野に多く使用され，私達の生活の中に深く浸透している．たとえば，テレビジョンはマイクロコンピュータの力によって美しい画面に自動的に最適同調され，ルームエアコンは温度や湿度を検知して快適な環境を最もエネルギーを節約した形で実現する．一方，後者は高機能・高性能ゆえに多くは産業分野において使用されている．制御，計測，通信，データ処理，工業などでの実績を踏まえて，最近の1つの特徴はOA，パーソナル コンピュータへの展開にあり，徐々に私達の生活に密着した分野に広まってきているといえる．特にパーソナル コンピュータは人間との対話が中心であるため，マイクロコンピュータのトータル パフォーマンスや価格の他にハード，ソフト両面の使いやすさが重要課題となる．また，パーソナル コンピュータの普及による実績と技術の蓄積は次の時代にくるであろうホーム コンピュータ普及への基盤を創生するものと思われる．

従来，パーソナル コンピュータは高性能8ビット マイクロコンピュータが使用されてきたが，最近は8ビットから16ビット マイクロコンピュータにその対象機種が移りつつあり，ここ1〜2年のうちに16ビット マイクロコンピュータに多くが移行するものと予

想される.

さて，現在16ビットマイクロコンピュータは68000，8086，Z8000など数機種が世界の主流を占めているが，なかでも68000は現在入手し得る16ビットマイクロコンピュータの最も高性能な機種であり，その能力の高さと使いやすさにより，今後広く普及するものと思われる.

本書は16ビットマイクロコンピュータ68000のハードウェアからソフトウェア，さらにシステム開発サポートまでを含めて解説したものである．68000を使用する技術者が，その内容と使い方を容易に理解し，実践できるように工夫した．本書は大きく分けてⅢ編から成る．第Ⅰ編（1章～10章）はハードウェアの解説である．ハードウェアはバスインターフェイス，入出力インターフェイス，例外処理，DMA転送技術などについて特に注意を払った．第Ⅱ編（11章～19章）はソフトウェアの解説である．ソフトウェアはできるかぎり多くの例題を取り入れることにより理解を深めることに留意した．第Ⅲ編（20章～22章）は開発サポートに関する解説である．全編を通じて，68000を使用する技術者，学生の立場に立って，例題を利用しつつ疑問点に対する解釈という視点で執筆を行った．本書が16ビットマイクロコンピュータ68000使用技術者の座右の書として役立つことを願っている.

最後に，本書刊行に当って御指導いただいた筑波大学 森亮一教授，日立製作所半導体事業部 初鹿野凱一氏，同 武蔵工場 中野浩行氏，石川知雄氏，ならびに丸善株式会社出版部各位に対し深謝の意を表する.

1983年2月

著　　者

執　筆　者

喜田祐三	株式会社 日立製作所
萩原吉宗	〃
岩崎一彦	〃
中村英夫	〃
塚田敏郎	〃
船橋恒男	〃
山口　昇	〃
高木克明	〃
沢瀬照美	〃
松浦達治	〃
野口孝樹	〃
志村隆則	〃

目　　次

I　68000 のハードウェア

II　68000のソフトウェア

I　68000のハードウェア

1 まえがき

1.1 マイクロコンピュータの進歩

マイクロコンピュータ 4004 が発表されてから，すでに 10 年が過ぎた．この間，LSI 技術の進歩，特に MOS-LSI の高集積化，高性能化はマイクロコンピュータの機能，性能を大きく発展させた．図 1.1 は 10 年間のマイクロコンピュータの進歩を，1 個のチップ上に集積化された回路規模すなわち MOS トランジスタの数で示したものである．初期のマイクロコンピュータ 4004 は，アルミゲートの P チャネル構造の MOS-LSI 技術で実現されていた．この LSI 技術は，集積度も低く，トランジスタの動作速度も遅かったので，内部の演算データのビット幅を 4 ビットとして回路規模を小形化し，また命令実行時間も数十 μs という低性能なマイクロコンピュータであった．

マイクロコンピュータの性能を飛躍的に改善するきっかけとなったのは，シリコンゲートの N チャネル構造 MOS-LSI 技術の出現であった．この技術により，内部演算データのビット幅が 8 ビット，命令実行時間が数 μs というマイクロコンピュータが実現できた．6800，8080，Z-80 などが代表的な機種として広く用いられている．

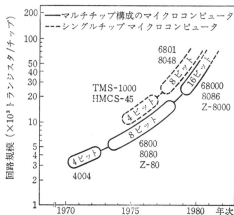



図 1.1 マイクロコンピュータの進歩

一方この時期に，演算回路，メモリ，入出力回路を 1 個のチップ上に含んだ 4 ビットシングルチップマイクロコンピュータが現

れてきた．このマイクロコンピュータは，演算性能は比較的低いが，チップ上に多くの機能を盛り込んだLSIである．このようにマイクロコンピュータは，6800のような高性能マルチチップ マイクロコンピュータとTMS-1000のような多機能シングルチップ マイクロコンピュータの2極化の傾向が現れてきた．これらは産業界のあらゆる分野で使用され，マイクロコンピュータの普及の礎を築いたともいえる．

LSI技術の最近の進歩により，LSI内部の配線の最小幅は3μmにまで微細化できるようになった．その結果さらに数倍～十数倍の集積度，スピードの向上を図れるようになった．このようなLSI技術を用いることによって16ビットの演算データ幅をもち，性能，規模で従来のミニコンピュータに匹敵するようなマイクロコンピュータが実現できるようになった．すでに世の中で使用されている16ビット マイクロコンピュータの代表的なものとしては，68000，8086，Z8000があげられる．

16ビット マイクロコンピュータは，性能，回路規模のみでなく，コンピュータ アーキテクチャも，8ビット時代のものに比べ大きく進歩している．

これらの特徴のいくつかを以下に示す．

a．メモリ アドレス空間の拡大　1メガバイト以上のメモリ アドレス空間をもっている．その結果，より複雑，大規模なプログラムが搭載され，主に高級言語によるプログラム開発が行われるようになる．

b．パフォーマンスの向上

（1） データ転送，演算の高速化のために，内部データ処理は最大32ビット単位で行われる．

（2） クロックの高速化とパイプライン方式などの先行制御を行っている．

（3） 乗算，除算，ストリング処理，ビット処理などの多機能命令の強化を図っている．

（4） アドレス形式の種類の増加と多機能化により，命令のスループットを向上させている．

（5） レジスタおよびスタックの機能の向上，数の増加により，処理効率を向上させている．

c．アーキテクチャの向上

（1） マイクロプログラミング制御方式，モジュール構造化等の設計技術を導入している．

（2） マルチプロセッサ システムを可能とするような命令，制御信号などを設けてい

る.

(3) より大規模なシステムを効率良く動作させるために，非同期バス インターフェイスをもっている.

1.2 68000 の 概 要

16ビットマイクロコンピュータ68000は，3 μm 加工のシリコン ゲートNチャネルMOS技術を用いて実現され，6.44 mm×7.26 mm のシリコン チップ上に約68000個のトランジスタを含んでいる．初期の8ビット マイクロコンピュータ6800と比べると，その性能，回路規模は図1.2に示すように大幅に向上してしいる．性能では約10倍，回路規模では約15倍となっている．さらにLSI技術の進歩を十分に活用することによって，将来に向けた新しいマイクロコンピュータ アーキテクチャの導入を図っている．命令体系には，8ビット マイクロコンピュータ6800との共通性はない．その結果，旧LSI技術によって実現された8ビット マイクロコンピュータのアーキテクチャにとらわれることなく，斬新なアーキテクチャを導入することが可能となった．68000は，最新の半導体技術，回路技術，計算機技術を駆使し，最高の性能と柔軟性，信頼性，使いやすさを追求して得られたマイクロコンピュータといえる．さらに，それは将来の32ビット マイクロコンピュータへの発展を考慮した構造をもつ16ビット マイクロコンピュータである.

その特徴の概要は次のとおりである.

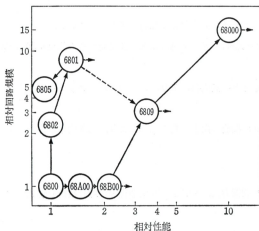



図 1.2 マイクロプロセッサの性能比較

（1） 一貫性と規則性をもったアーキテクチャ

（2） 多機能，高性能なプロセッサ構造

（3） 効率的なプログラミング手法のサポート

（4） ソフトウェアの信頼性向上

（5） 高機能な例外処理

（6） マルチプロセッサ用機能

（7） 多機能インターフェイス

本書はⅢ編に分れている．Ⅰ編では，68000のハードウェアについて述べる．そこでは，アーキテクチャ，インターフェイス信号，アドレス方式，命令，例外処理，周辺デバイスとのインターフェイス，68000周辺LSIについて記述している．ここで使用している用語等は，極力モトローラ社あるいは日立製作所の説明書と同じになるように努めた．

Ⅱ編では，68000のソフトウェアについて述べる．そこでは，アセンブラ，各命令グループの詳細な説明，割込み処理などを述べる．各命令の説明時に，必要に応じて短いプログラムの例をあげているが，最後に命令の使い方をより理解してもらうために，6種のプログラム例を記載した．

Ⅲ編はサポートシステムである．ハードウェアおよびソフトウェアのサポートシステムについて述べる．紙面の都合で，詳細な説明は割愛した．ハードウェアサポートシステムは日立製作所の製品について紹介する．ソフトウェアサポートシステムも，日立製作所から提供されているものを紹介した．16ビットマイクロコンピュータシステムにおいて，今後益々重要となるオペレーティングシステムについては，本マイクロコンピュータシリーズの別版として発刊されることを期待して，本書ではふれていない．

付録として，アドレス形式，および命令一覧表を載せ，本文理解の一助となるよう配慮してある．

2　68000のアーキテクチャ

マイクロプロセッサ68000は，最新のVLSI（Very Large Scale Integrated Circuit）技術と計算機技術により，高密度，高速性，高いフレキシビリティ，信頼性および使いやすさを実現した高性能16ビット マイクロプロセッサである．そのアーキテクチャは，マイクロプログラム制御方式の導入による斬新なLSI構造を有するとともに，将来の32ビット マイクロプロセッサへの展開を考慮したものとなっている．

2.1　アーキテクチャの特徴

本節では，68000 MPU（Micro-Processing Unit）のアーキテクチャの特徴について簡単にふれるが，その詳細説明は，本章の次節以後および3章以後で行うこととする．

68000アーキテクチャの特徴は次のとおりである．

（1）　レジスタが多く，使いやすい．

32ビットの多機能レジスタを17個もつ．そのうち8個はデータ用，9個はアドレス用である．9個のアドレス用のレジスタのうち2個はシステム スタック ポインタとして使用する．

（2）　アドレス空間が広い．

16メガバイトの物理（フィジカル）アドレス空間をもつ†．さらに64メガバイトまでアドレス空間を広げる手段を有する．

（3）　命令は，レギュラな体系で構成され，機能が豊富である．

56種の基本命令をもち，高級言語をサポートする命令を含む．

---

† 物理アドレスは，ハードウェアで定まるアドレスと定義する．これに対し論理アドレスは，ソフトウェア上で定義しているアドレスである．

（4） アドレス形式が強力である．

6種の基本モードがあり，そのなかに計14種のバリエーションをもつ．

（5） 種々のタイプのデータの演算が可能である．

1, 4, 8, 16, 32各ビット長のデータを扱える．

（6） 強力なベクタ割込み機能をもつ．

（7） マルチプロセッサ システム用のハードウェア，ソフトウェア制御機能をもつ．

（8） スループットの高いバス インターフェイスをもつ．

同期および非同期方式のバス インターフェイスをもつ．アドレス バスは24ビット，データ バスは16ビットで構成されておりマルチプレックスされていない．また8ビット マイクロプロセッサの周辺LSIと直結できる．

（9） システムの信頼性向上，およびテスト効率向上を図る機能をもつ．

プログラム実行状態をスーパバイザ状態とユーザ状態に分け，システムの信頼性向上を図っている．またトラップ，トレース機能を有し，システムのテスト効率を向上させている．

（10） メモリマップドI/O方式である．

周辺LSIなどに対しても，すべてのメモリ アクセス命令を有効に使用できる．

以上のように，68000はミニコンピュータおよび大形コンピュータのアーキテクチャの設計思想を取り入れており，広い分野への応用を可能とするような，フレキシブルで，汎用的な構造を有している．

2.2 プロセッサの内部構造

68000は，以下に述べるようなマイクロプログラム制御方式を取り込んだ新しいタイプのVLSIアーキテクチャをもっている．図2.1に示すように，内部は大きく実行ユニットと命令制御部に分けられる．実行ユニットは，レジスタと演算回路で構成されている．これは，データおよびアドレスの演算を行う部分である．実行ユニットでは，アドレスおよびデータ用の2個の16ビット幅のバスで各部を結合している．各バスごとに並列にデータ転送ができるので，LSI内部での各レジスタ，演算回路間のデータ転送，演算の効率が向上する．バス構造のもう1つの特徴は，各バス上の2箇所にスイッチがある点である．これらのスイッチを制御することにより，互いに独立したセクションとしてバスを分離できる．分離された各セクション毎に上位ワード，下位ワードの転送を同時に行うことがで

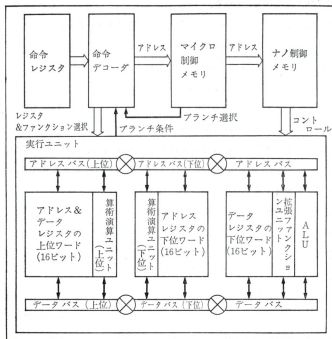



図 2.1　68000の内部構造

きるため，内部のデータ転送効率を向上させることができる．さらに，3個の演算制御ユニットでは，アドレス演算およびデータ演算が並列に行われる．

特に拡張ファンクションユニットは，32ビット演算のためのシフト機能，ビット操作機能などの特別な論理操作を行い，ALU (Arithmetic Logic Unit) を補完する．以上の実行ユニットに隣接して，以下に説明する制御用の各ブロックが配置されている．

68000の制御は，前述したようにマイクロプログラム制御方式を採用している．マイクロプログラム制御方式は，規則的なメモリ構造を利用して複雑な制御回路を実現できることが特徴である．またVLSIチップの開発時に，マイクロプログラム用メモリの書換えにより制御シーケンスが容易に設定でき，フレキシビリティに富むため，複雑なVLSIチップの設計法としてよく使われている．次に，このマイクロプログラムの動作を簡単に説明する．

オペレーションコードはLSIチップの外部のメモリから入力され命令レジスタに取り

込まれた後，図2.1に示されているように，命令デコーダで解読される．その結果，実行ユニットへはレジスタおよびファンクションの選択信号が出力され，マイクロ制御メモリへはマイクロ命令シーケンス起動のためのアドレスが出力される．68000のマイクロプログラム制御方式の特徴は，1個のLSIチップに全機能を効率良く集積化するために，2レベルの制御構造を採用し制御用メモリのサイズの小形化を図っていることである．第1のレベルは，ビット幅の短いマイクロ制御命令とよばれる命令形式で，複雑なブランチ能力により，マイクロプログラムのシーケンスを制御する．第2のレベルは，ビット幅の広いナノ制御命令とよばれる命令形式で，実行ユニットの動作を制御する．これらの制御シーケンスは，図2.1に示すように，それぞれ別のメモリ（ROM: Read Only Memory）に記憶されている．LSI外部から取り込んだ命令によりマイクロ制御命令のシーケンスが起動されると，まずマイクロ制御命令からナノ制御メモリのアドレスが出力される．その出力アドレスによりナノ制御命令が読み出される．このナノ制御命令が直接に実行ユニットを制御することとなる．

2.3 レジスタの構成

図2.2は，68000のレジスタ構成を示す．68000のレジスタは8個のデータ レジスタ

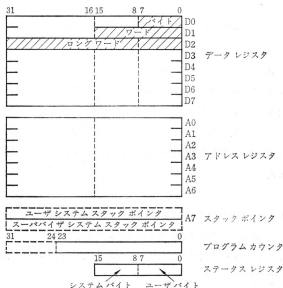



図 2.2 レジスタ構造

(D0~D7), 7個のアドレス レジスタ (A0~A6), 2個のシステムスタック ポインタ, 1個のプログラム カウンタおよび1個のステータス レジスタで構成される.

68000は, 構造的に統一性の高い16ビットのマイクロプロセッサである. 特にレジスタ, データに関して次のような特徴をもつ.

(1) データ レジスタは完全に対称である. すなわちどのデータ レジスタも, オペレーションのオペランド (許されるアドレス形式内で) となり得る.

(2) すべてのデータ レジスタとメモリ内のデータは, ビット (1ビット), ディジット (4ビット), バイト (8ビット), ワード (16ビット) およびロングワード (32ビット) の演算が可能である.

(3) バイト, ワード, ロングワードの各オペランドは, 整数演算を受ける. オペランドサイズは命令によって指定される.

(4) すべてのアドレス レジスタは, データをアドレッシングするために同等に使うことができる.

(5) アドレス レジスタのデータは, ワード, ロングワードのアドレス演算ができる.

各レジスタ群について, 以下に説明する.

2.3.1 データレジスタ (D0~D7)

8個のデータ レジスタはそれぞれ32ビットの構成である. オペランドの指定により, これらをバイト レジスタ, ワード レジスタ, ロングワード レジスタとして扱うことができる.

図2.2に示すように, バイトの場合は下位8ビット, ワードの場合は下位16ビット, ロングワードの場合は32ビット全体を用いる. LSB (Least Significant Bit; 最下位ビット) はビット番号0, MSB (Most Significant Bit; 最上位ビット) はビット番号31として表される. データ レジスタがバイトおよびワードオペレーションのデスティネーションオペランドとして用いられるときは, 下位部のみ変化し, 残りの上位部は変化しない†.

2.3.2 アドレスレジスタ (A0~A6), システムスタックポインタ (A7)

アドレス レジスタとシステムスタック ポインタは32ビット長で構成されている. これらはバイト サイズで用いることはできない. したがってアドレス レジスタがオペレーシ

† MOVEQ, MOVEM命令は例外

ョンのソース オペランドとして用いられるときは，オペレーション サイズに応じて下位のワードか，ロング ワードがオペランドとして用いられる．アドレス レジスタがデスティネーション オペランドとして用いられるときは，オペレーション サイズに関係なく，そのレジスタの全ビットが影響を受ける．オペレーション サイズがワードならば，オペランドは演算実行前に，その15ビット目（符号ビット）が16ビットから31ビットまでの上位の全ビットに自動的にコピーされ演算される（符号拡張：12.1節で述べる）．

なおデータ レジスタ，アドレス レジスタは，共にインデックス レジスタとして用いることも可能である．

A7はアドレス レジスタとして用いられるのみでなく，システム スタック ポインタにもなっている．ここでスタック† とは，データの後入れ，先出し（Last-In-First-Out）リスト構造のレジスタ群（メモリ）のことである．A7は割込みのときなど，システムスタック ポインタとしてプロセッサが自動的に内容を変化させる場合があるので，プログラム スタック ポインタとして使用するときには注意しなければならない．A7をシステム スタクッ ポインタとして用いるときには，アドレスを自動的に増加して，新しいメモリ アドレスにデータ(戻り番地，例外処理時の退避データ等)を書き込んだり，自動的に減少して，すでにデータが書き込まれているアドレスからデータを読み出したりする機能をもつ．

スタック ポインタは，図2.2のA7に示すように，ユーザ状態のときと，スーパバイザ状態のときで，自動的に切り換えて使用される．すなわちA7のレジスタは，ユーザ プログラム状態のときは，ユーザ システム スタック ポインタ（USP）となり，スーパバイザ プログラム状態のときは，スーパバイザ システム スタック ポインタ（SSP）となる．68000がスーパバイザ状態であるとき，A7のレジスタがスーパバイザ システム スタック ポインタとなっているから，ユーザ システム スタックポインタはアドレス レジスタとして参照することはできない．また逆も同様である．

---

† 68000のマニュアル等では，システム スタックとユーザ スタックという各称で2つのタイプのスタックを定義している．後者はアドレス レジスタ A0～A6 および A7を用い，命令のアドレス レジスタ間接モードのポスト インクリメント（実行後アドレス レジスタに1を加える）およびプリデクリメント（実行前にアドレス レジスタから1を減ずる）の各操作を行うことにより，プログラム的にスタックを構成するものである．これに関しては12章でプログラムの一例として示す．

本書ではA7をスタック ポインタとして用いるスタックとA0～A6およびA7をアドレス レジスタとして用いるスタックを次のように区別することとする．

- スタック
  - システムスタック（A7を用いる）
    - スーパバイザ システム スタック
    - ユーザ システム スタック
  - プログラム スタック（A0～A6およびA7を用いる．ただしA7はシステム スタックとして用いるのでユーザは使用をさけたほうがよい）

（a）ユーザ システム スタック (S=0)
(サブルーチン ジャンプの退避例)

（b）スーパバイザ システム スタック (S=1)
(割込み時の退避例)

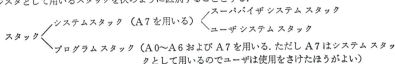
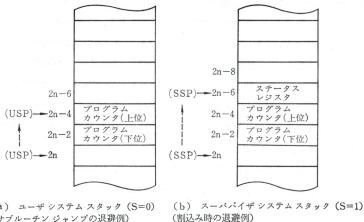



図 2.3 システム スタック

これらのシステム スタック ポインタにより，図2.3に示すようなユーザ システムスタック（S†=0のとき）とスーパバイザ システム スタック（S=1のとき）を実現できる.

図2.3に示すように，これら2つのシステム スタックはメモリ アドレスの大きい方から小さい方へ順に使用される．すなわちシステム スタックにデータがプッシュされるとスタック ポインタは減算され，システム スタックよりデータが取り出されるとスタック ポインタは増加する.

サブルーチン呼出しの際は，プログラム カウンタがア クティブなシステム スタックに退避され，リターン時にアクティブなシステム スタックから復帰される．一方トラップ処理，割込み処理などの例外処理が起動されるとスーパバイザ状態となり，プログラム カウンタ，ステータス レジスタなどをスーパバイザ システム スタックへ退避する.

システム スタックへの退避情報についての詳細は8章にて説明する.

システム スタック上のデータ配列を適切に行うために，システムスタックに対する データエントリのアドレスはワードの境界に限定され，データは常にワードの境界単位で格納される.

† Sは後述するステータス レジスタ内のフラグである.

2.3.3　プログラム カウンタ（PC）

68000は32ビットのプログラム カウンタをもつが，現在はその下位24ビットのみをLSIチップ外に出力している．したがって生成できるアドレスは，16進表示で$000000から$FFFFFF（10進表示では，0から16 777 215）の範囲となる．命令長がワード単位であるため命令は偶数番地で示されるアドレスに格納されなければならない．このためプログラム カウンタには必ず偶数アドレスを設定しておくことが必要である．奇数アドレスを設定した場合はアドレス エラーが発生する．このプログラム カウンタの1ビットから23ビットがチップ外に出力され，A1～A23のアドレスバスとなっている．

2.3.4　ステータス レジスタ

ステータス レジスタは図2.4に示すように，16ビットで構成され，ユーザ バイトとシステム バイトに分けられている．図2.4にステータス レジスタのビット構成を示す．

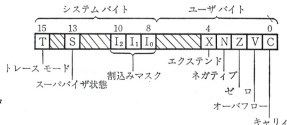



図 2.4　ステータス レジスタ

a．ユーザ バイト　ユーザ バイトには，キャリィ（C），オーバフロー（V），ゼロ（Z），ネガティブ（N），エクステンド（X）のコンディション コード フラグが割り当てられている．

b．システム バイト　システム バイトは，次の3種のフラグで構成される．

（i）**割込みマスク**（I0～I2）　3ビットで，現在実行中のプログラムの優先順位(プライオリティ レベル：0～7）を指定する．ここで示されたレベルと同じかまたは低いレベルの割込み要求は受け付けられない．ただしレベル7の割込みは常に受け付けられる．

（ii）**スーパバイザ状態フラグ**（S）　実行プログラムがスーパバイザ状態（S=1）であるか，ユーザ状態（S=0）であるかを示す．

（iii）**トレース モード フラグ**（T）　本ビットが“1”にセットされていると，命令実行ごとにトレース例外処理を行う．すなわち命令を実行するたびにスーパバイザ状態に入

り，ユーザのトレース サービス ルーチンへジャンプする．

2.4 データの構成

2.4.1 メモリ内のデータの構成

68000のシステムにおけるメモリ内のワードおよびバイト構成を図2.5に示す．1ワードを構成する2バイトのうち，上位バイトのアドレスはワード データのアドレスと同じ偶数アドレスとなっている．下位バイトはワード データのアドレスより1だけ大きい奇数アドレスになっている．これらバイト データは個別にアクセスすることができる．

図 2.5 メモリのワード／バイト構成

命令（1ワード以上で構成される）および複数バイト（ワード，ロング ワード）データは，ワードの境界ごと（偶数アドレス）にアクセスされる．ロング ワード データが偶数アドレスnに格納されていると，次のデータはアドレスn+4に格納されることとなる．

68000は，ビット データおよび8ビット，16ビット，32ビットの整数データ，ならびに32ビットのアドレス データ，2進化10進数（BCD；Binary Coded Decimal）表現のデータの演算を行う．これら各形式のデータがメモリ内に格納されるときのアドレス配置を図2.6に示す．図中でハッチした部分が命令によって演算される単位である．2進化10進数データは1バイトの中に2ディジット（1ディジット：4ビットで構成される）を含むように構成されている．10進演算はバイト単位で行われるので，2ディジット分を同時に演算する．

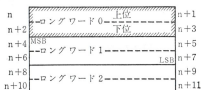
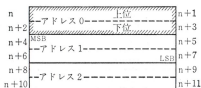



図 2.6 メモリ内の演算データの構成

2.4.2 整数データの表現形式

68000では2値数，2進化10進数および2の補数表現による2進数のデータ形式を扱う．ここで2の補数表現について説明する．バイト，ワード，ロングワードの整数を2の補数(2's Complement) 表現で表すと図2.7のようになる．各サイズのデータは，最上位ビットに符号ビットを配置している．したがってバイトデータは −128($80)〜+127($7F)，ワードデータは−32 768 ($8000)〜+32 767 ($7FFF)，ロングワードは −2 147 483 648 ($80000000)〜+2 147 483 647 ($7FFFFFFF) の範囲のデータを表現できる．

68000では，バイトからワードまたはロングワードへ，ワードからロングワードへとデ

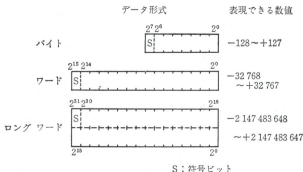



図 2.7 整数データの表現形式（2の補数表現の場合）

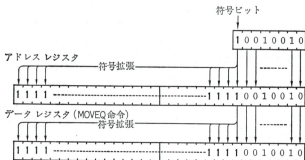



図 2.8 符号拡張（バイト→ロング ワードの例）

ータ サイズを拡張できる．このときの整数データの符号の統一を図るために，符号拡張機能をもっている．

データ サイズの拡張時における符号拡張機能について図2.8に示す．この図ではバイト データからロング ワード データへの拡張の例を示す．符号拡張を行うには，ソース データの符号ビットの内容が，そのビット番号より大きなビット位置にコピーされる．符号ビット以外の各ビットの内容はそれぞれの対応したビット番号の位置へ移る．この符号拡張は各データの演算の前に実行される．

アドレス レジスタがデスティネーション レジスタとして指定される場合は，常に符号拡張が行われる．

データ レジスタがデスティネーション レジスタとして指定される場合は，MOVEQ および MOVEM の命令を実行したときのみ符号拡張が行われる．

2.5 基本システム構成と動作

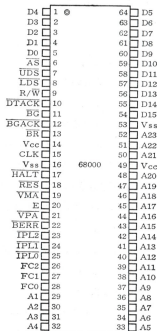
2.5.1 入出力インターフェイスの構成と特徴

68000 の LSI チップは，図 2.9 に示すような 64 ピンのデュアル イン ライン（Dual In Line）形のパッケージに搭載されている．本章では入出力インターフェイスの構成の概要と特徴について述べる．

図 2.9 ピン配置

a．非同期パラレル バス　非同期バス制御ラインには，アドレス ストローブ（$\overline{\mathrm{AS}}$），リード/ライト制御（R/$\overline{\mathrm{W}}$），上位データ ストローブ（$\overline{\mathrm{UDS}}$），下位データ ストローブ（$\overline{\mathrm{LDS}}$）およびデータ アクノレッジ（$\overline{\mathrm{DTACK}}$）がある．これらの制御信号によって，メモリ，周辺 LSI などデータ アクセス時間の異なる外部デバイスとのデータ転送効率を向上させる．68000 MPU はメモリ，周辺 LSI との間でデータを転送を行うときに，相手からデータ転送が完了することを知らせる信号（$\overline{\mathrm{DTACK}}$）を入力することによって，データ転送サイクルを終了させる．

b．マルチプロセッサ用のバス インターフェイス　同一バス上にバスを支配できるデバイス（バス マスタ）が複数個存在する場合（たとえば複数個の MPU または DMAC），そのバスの支配権のやりとりが，バス要求（$\overline{\mathrm{BR}}$），バス グラント（$\overline{\mathrm{BG}}$），バス グラント アクノレッジ（$\overline{\mathrm{BGACK}}$）信号によって，ハンドシェーキング方式で行われる．この一連の動作がバス アービトレーションとよばれる．

c．非マルチプレックス バス　マイクロプロセッサのような LSI では，パッケージのピン数の制約から，1 本の入出力信号線に複数の機能をもたせる場合がある．特にメモリのアドレス空間を広くとるために，アドレスとデータを時分割で同一バス上にのせる方法が採用されることがある．しかしこの方法ではデータの転送効率を低下させる欠点があるため，68000 では 23 ビットのアドレス バス A1～A23 と $\overline{\mathrm{UDS}}$, $\overline{\mathrm{LDS}}$ 信号および 16 ビ

ットの双方向データバスD0〜D15を別々に設けている．これによって高速なバスサイクルを実現している．

d. メモリマップドI/O　メモリ，周辺LSIなどのすべての外部デバイスを単一バス上に結合し，コミュニケーションを行う．すなわち同一メモリアドレス空間に，メモリ，周辺LSIのアドレスを割り付ける．これによって周辺LSIに対しても，68000のすべての命令，アドレス形式が有効に利用できる．

e. 優先割込み構造　68000には3本（$\overline{\text{IPL}0}$，$\overline{\text{IPL}1}$，$\overline{\text{IPL}2}$）の割込み制御入力がある．これは割込みを要求するデバイスからの優先レベルをコード化した入力信号である．レベル0は割込み要求のないことを示し，レベル7が最優先のノンマスカブル割込み要求であることを示す．

この入力割込みレベルが，内部のステータスレジスタ†に格納されている割込みマスクレベルと比較され，それ以下のときは割込みは受け付けられない．

f. 8ビットマイクロコンピュータ6800ファミリのLSIと直結できるバスインターフェイス　68000は上記非同期バスインターフェイスのほかに，同期式のバスインターフェイス制御信号線をもつ．図2.9のイネーブル（E），バリッドメモリアドレス（$\overline{\text{VMA}}$），バリッドペリフェラルアドレス（$\overline{\text{VPA}}$）およびリード/ライト制御（R/$\overline{\text{W}}$）を同期式インターフェイスのために用いる．これによって68000のシステムに同期式のバスインターフェイスをもつ8ビットマイクロプロセッサ6800の周辺LSI（PIA，PTM，FDC，CRTC，ACIA，SSDA，ADLC，GPIA等）を利用することができる．

g. メモリマネージメント機能　3本のプロセッサステータス信号線（FC0，FC1，FC2）を介して，プロセッサのプログラム実行状態を知ることができる．これを用いてメモリ空間にユーザデータ，ユーザプログラム，スーパバイザデータ，スーパバイザプログラム領域を別々に設けることができる．その結果アドレス空間を64メガバイト（16メガバイト4セグメント）にまで拡張できる．これらの信号線を使うことによって，メモリ管理ユニットMMU（Memory Management Unit）によるメモリ保護，アドレスマッピング等を行うことができる．

h. バスエラー入力信号　入力信号$\overline{\text{BERR}}$はバスエラー発生を示す入力で，不応答デバイス，不当メモリアクセス，割込みベクタ番号取込み不良などを68000に知らせる

---

† 割込み制御入力は負論理であるが，内部のステータスレジスタの情報は正論理で表されている．

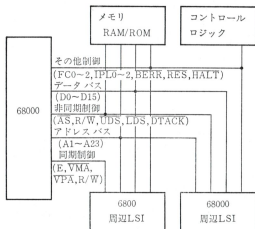



図 2.10 68000の基本システム接続構成

のに有効である．この信号によってバス サイクルを終了させ68000は例外処理の実行に遷移する．図2.10は，68000の基本システムの接続構成を示す．

2.5.2 基本動作サイクル

68000のサイクル タイムは次のように分けられる．

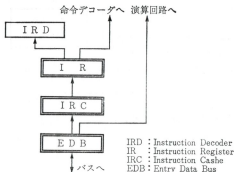
a. クロック サイクル 入力クロックの周期である．そのクロック パルスの立上りエッジから次のクロック パルスの立上りエッジまでの時間である．

b. バス サイクル バイト，ワード データの読出し，書込みを行うのに必要なタイミング シーケンス全体である．4クロック サイクルで最小バス サイクルが構成される．非同期バス インターフェイスで $\overline{\text{DTACK}}$ 信号が遅れて入力されるときは，本バス サイクルは4+n クロックの形で延長される．

c. 命令サイクル 1命令を実行するのに必要なタイミング シーケンスである．一般に複数のバス サイクルによって構成される．各命令の実行時間は，上記クロック サイクルの数で示される．

2.5.3 命令プリフェッチ機能

68000は高速性を実現するために命令プリェッチ（命令の先取り）とよばれる機能を有する．この機能はある命令を実行中に次の命令を MPU 中に取り込むもので，命令フェッチに要する時間を見掛上節約できるという利点がある．68000はこの機能を実現するため

図 2.11　命令プリフェッチ用レジスタ群

に図2.11に示すような3本のレジスタを有している．現在実行中の命令コードは IR (Instruction Register) にセットされ，命令がデコードされたのち，IRD (Insruction Decoder) に移される．命令が IRD に移されると次に実行される命令が IRC から IR に移されデコードの開始を待つ．原則として IR, IRC (Instruction Cashe) には次に実行される命令の第1ワード目が各々セットされる．命令の第2ワード目からの拡張部分は EDB (Entry Data Bus) にセットされる．このように，プリフェッチ用の特別なレジスタを有することにより次に実行される命令の全体または一部を先取りし，実効的に命令フェッチ時間を節約し得るだけでなく，IRD を設けることにより次の命令のデコードを早期に開始し高速化を図ることが可能となる．

シングル ワードの分岐命令 (Bcc, DBcc, JMP 命令等) を実行したとき，分岐が行われる場合は，プログラムの流れが飛ぶために，あらかじめフェッチ済の次のアドレスの命令ワードは不要となる．そのとき，新たに2ワードの命令をフェッチするような制御が行われる．68000では前の命令実行終了までに2ワードがプリフェッチされるため，アドレス空間の最後の1ワードにシングル ワードの分岐命令などを配置しておくと，前の命令の実行中に行われる命令フェッチ動作でアドレス エラーを発生するので注意が必要である．

2.5.4 基本命令シーケンス

図2.12に68000の基本命令シーケンスの一例を示す．前項で述べたプリフェッチ機能を有しているために，各命令の実行時には，引き続いて行われる命令列の2ワード分がメモリから MPU 内に取り込まれる．図にハッチで示したバス サイクルは MPU の演算結果のワード データをメモリへ転送するサイクルである．条件ブランチ命令 (Bcc) を実行するときは，2クロックの区間で条件判定を行う．この間はデータ バスは使用されない．前節で述べたように分岐が行われるときは，前のサイクルでフェッチされた命令ワードは捨てられ，分岐先の新命令を2ワード フェッチする．

分岐命令で分岐したときは新たに命令を2ワード フェッチするが，それ以外の通常命令

図 2.12 基本命令シーケンス

では，その命令自身のワード数分だけ新しい命令ワードをフェッチするようなシーケンスをもっている．

3　インターフェイス信号とバスオペレーション

この章では68000のインターフェイス信号の機能について説明し，次に，それらの信号の基本動作であるバスオペレーションの種類とタイミングについて述べる．

3.1　インターフェイス信号

68000のインターフェイス信号は図3.1に示すようなグループに機能的分類をすることができる．以下に各機能グループごとに信号の説明を行う．

3.1.1　アドレスバス，データバス

a.　アドレスバス（A1～A23）［出力］　68000のアドレスバスは23ビットで構成され，8メガワード（16MByte）のデータを直接にアドレッシングすることができる．ア

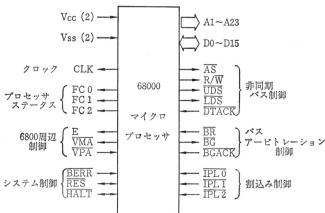



図 3.1　信号の機能グループによる分類

ドレス バスは単方向性のスリーステート バスで，割込みサイクルを除くバス オペレーション サイクルに対してアドレスを供給する．ここで供給されるアドレスはワード データ（16 ビット）をアクセスするためのもので，バイト データ（8 ビット）をアクセスするためには A0 なるアドレス信号が必要である．68000 におけるアドレス付けはバイトを単位として行われ，プロセッサ内部には A0 のアドレスが存在する．A0 のアドレスは直接には外部に出力されていないが，プロセッサがバイト データをアクセスするときには A0 の値に応じて上位データ ストローブ（$\overline{\text{UDS}}$）または下位データ ストローブ（$\overline{\text{LDS}}$）が出力される．これについては後の 3.1.2 項 c. で述べる．

アドレス バスは，割込みサイクルのときには別の意味をもつ．すなわち，割込みアクノレッジ サイクルではアドレス バスの A1，A2，A3 からはどのレベルの割込みがサービスされているかについての情報が供給される．なお，アドレス バス A4～A23 はこのときすべて "High" レベルにセットされる．

b. データ バス（D0～D15）［入出力］　データ バスは 16 ビットの双方向性のスリーステート バスで，プロセッサが外部のデバイス（メモリや周辺装置）とデータの授受を行うための汎用のデータ通信路である．データ授受はワード長（16 ビット）またはバイト長（8 ビット）のいずれでも可能である．

データ バスを介してプロセッサと外部デバイスの間で授受されるデータは，通常，プロセッサを動作させるための命令およびデータである．ただし割込みアクノレッジ サイクルのときだけは，外部デバイスからプロセッサに対して例外ベクタ番号が転送される（プロセッサはこのときにバス上に乗っているデータを例外ベクタ番号として処理する）．

3.1.2 非同期バス制御信号

以下の信号はプロセッサが外部デバイスとデータの授受を非同期的に行うための信号である．

a. アドレス ストローブ（$\overline{\text{AS}}$）［出力］　アドレス ストローブはアドレス バスに有効なアドレスが出力されていることを示す信号である．

b. リード/ライト（R/$\overline{\text{W}}$）［出力］　この信号はデータ バス上のデータ転送がリード サイクルなのかライト サイクルなのかを定める．すなわち，R/$\overline{\text{W}}$＝"High" のときリード サイクル，R/$\overline{\text{W}}$＝"Low" のときライト サイクルである．

c. 上位データ ストローブ，下位データ ストローブ（$\overline{\text{UDS}}$, $\overline{\text{LDS}}$）［出力］　データ ストローブ信号はプロセッサがデータ バス上のデータを読み取るタイミングまたはデー

タバス上へデータを出力するタイミングを示すものである．データストローブ信号は本来は１本だけでもすむが，68000の場合は $\overline{\mathrm{UDS}}$, $\overline{\mathrm{LDS}}$ と２本用意することにより，メモリのワードだけでなくバイトごとのリード/ライトオペレーションが可能となっている．

表3.1に $\overline{\mathrm{UDS}}$, $\overline{\mathrm{LDS}}$ の働きを示す．この表からわかるように，$\overline{\mathrm{UDS}}$, $\overline{\mathrm{LDS}}$ がインアクティブ†であるかぎりリード/ライトオペレーションは行われない．また，ワードのリード/ライトオペレーションでは $\overline{\mathrm{UDS}}$, $\overline{\mathrm{LDS}}$ のいずれもがアサートされ，バイトのリード/ライトオペレーションでは内部のアドレスA0の“Low”, “High”に対応して，それぞれ $\overline{\mathrm{UDS}}$, $\overline{\mathrm{LDS}}$ の一方だけがアサートされる．

表 3.1　データバスのデータストローブ制御

R/$\overline{\mathrm{W}}$	UDS	LDS	A0	D8〜D15	D0〜D7	オペレーション
—	High	High	—	****†1	****†1	No operation
High	**Low**	**Low**	—	D8〜D15	D0〜D7	ワードリード
High	High	**Low**	High	****†1	D0〜D7	下位バイトリード
High	**Low**	High	**Low**	D8〜D15	****†1	上位バイトリード
Low	**Low**	**Low**	—	D8〜D15	D0〜D7	ワードライト
Low	High	**Low**	High	D0〜D7†2	D0〜D7	下位バイトライト
Low	**Low**	High	**Low**	D8〜D15	D8〜D15†3	上位バイトライト

†1　****は無効データ
†2　D8〜D15にはD0〜D7と同じ内容が出力される．
†3　現状の68000ではD0〜D7にはD8〜D15と同じ内容が出力される．

d.　データ転送アクノレッジ（$\overline{\mathrm{DTACK}}$）［入力］　この信号は外部デバイスがデータをリードまたはライトすることが可能になったタイミングをプロセッサに知らせるための信号である．後で述べるようにプロセッサのリードまたはライトのバスサイクルはS0〜S7の８つの状態で構成され，プロセッサはS4の後縁で外部から $\overline{\mathrm{DTACK}}$ がアサートされているか否かをチェックする．もし，このとき，$\overline{\mathrm{DTACK}}$ がアサートされていなければ，S4とS5の間にSw（ウエート状態）が挿入され，プロセッサは $\overline{\mathrm{DTACK}}$ がアサートされるまで待つことになる．すなわち，アクセスタイムの遅い外部デバイスも，そのアクセスタイムに応じて $\overline{\mathrm{DTACK}}$ を遅らせてアサートすれば，プロセッサとのデータ授受を確実に行うことができる．したがって，外部デバイスとプロセッサ間の非同期の

---

† 68000のマニュアルには，信号がアクティブまたは真（true）であることを示すためにアサート（Assert）という言葉を用い，インアクティブ（休止状態）か偽（False）であることを示すためにネゲート（Negate）という言葉を用いている．しかし本書では従来からの慣例からみて，アクティブ，インアクティブの方が，各信号の電圧の“Low”, “High”のいかんにかかわらず，実際の動作をよく表現していると考えられる場合は，後者の用語を用いた．

データ転送を可能にしているのは，この $\overline{\text{DTACK}}$ 信号である．

3.1.3 バス アービトレーション制御信号

以下の3つの信号は，プロセッサも含めて2つ以上のバス マスタになり得るデバイスが存在するときに，どのデバイスがバス マスタになるかを決定するのに必要な信号である．

a. バス要求（BR）［入力］　この信号は，外部デバイスがバス マスタになることをプロセッサに要求する信号である．この信号の入力端子には，バス マスタとなり得るすべてのデバイスからの信号のワイヤド オアがとられたものが入力される．

b. バス グラント（$\overline{\text{BG}}$）［出力］　この信号は，プロセッサが他のすべてのバス マスタとなり得るデバイスに対してバスの解放を行うことを示す信号である．バスの解放はそのとき実行中のバス サイクルが終了した時点で行われる．

c. バスグラント アクノレッジ（$\overline{\text{BGACK}}$）［入力］　この信号は，外部のデバイスがバス マスタになったことを示す入力信号である．外部デバイスは以下の4つの条件を検出するまではこの信号をアサートしてはならない．

（1） バス グラント信号（$\overline{\text{BG}}$）を受信すること

（2） アドレス ストローブ信号（$\overline{\text{AS}}$）が休止状態（インアクティブ）であること．つまり，プロセッサがバスを使用していないこと

（3） データ転送アクノレッジ信号（$\overline{\text{DTACK}}$）が休止状態であること．つまり，メモリまたは入出力デバイスがバスを使用していないこと

（4） バス グラント アクノレッジ信号（$\overline{\text{BGACK}}$）が休止状態であること．すなわち，バス マスタ権をとっている外部デバイスが他に存在しないこと

3.1.4 割込み制御信号

割込み優先 レベル（$\overline{\text{IPL0}}$，$\overline{\text{IPL1}}$，$\overline{\text{IPL2}}$）［入力］　これら3本の割込み制御信号は外部デバイスがプロセッサに対して割込み要求があることを示すために用いられる．外部の割込みは3ビットにエンコードして入力され，7レベルの割込みが可能である．割込みレベルは7が最も優先度が高く，1が最も優先度が低い．また0は割込みがないことを示す．ここで，$\overline{\text{IPL0}}$ が最下位ビット（LSB），$\overline{\text{IPL2}}$ が最上位ビット（MSB）である．

3.1.5 システム制御信号

a. バス エラー（$\overline{\text{BERR}}$）［入力］　この入力信号は，現在実行中の命令サイクルで障

害が生じたことをプロセッサに知らせるための信号である．ここでいう障害とは次のようなものである．

（1） デバイスが応答しないとき

（2） 割込みベクトル番号をとらえられないとき

（3） 不正なアクセス要求がメモリ管理ユニット（MMU）によって検出されたとき

（4） 他に応用システムに依存するいろいろなエラーが発生したとき

b. リセット（$\overline{\mathrm{RES}}$）［入出力］　この双方向信号線は，外部のリセット信号に応じてプロセッサをリセットする働きをもっている．すなわち，この端子に“Low”レベルが入力されると，プロセッサはシステム イニシャライズ シーケンスを開始する．また，プロセッサの内部で RESET 命令が実行されると，この端子よりリセット信号が出力される．すなわち，プロセッサは外部デバイスをリセットすることができる．ただしこの場合プロセッサ自身の状態はリセットされない．なお，プロセッサおよび外部デバイスを含めた全システムをリセットするには，リセットとホールト端子に同時に Low レベルを入力する必要がある．

c. ホールト（$\overline{\mathrm{HALT}}$）［入出力］　この双方向信号線へ外部から“Low”レベルを入力すると，プロセッサは現在のバスサイクルを完了した時点で停止する．プロセッサが停止すると，すべての制御信号は休止状態になり，データ バスおよびアドレス バスはハイインピーダンス状態になる．また，プロセッサが2重バス障害のような状態になったときもプロセッサは停止し，このことを外界に知らせるため，この端子より“Low”レベルが出力される．

3.1.6 6800周辺 LSI 制御信号

以下の3つの信号線は，68000が6800の周辺 LSI を使用することを可能にするために設けられたものである．

a. イネーブル（E）［出力］　この信号は6800周辺 LSI に共通なデータ転送の同期クロック信号である．このクロックの周期は68000の10クロック サイクルに等しく6クロックが“Low”，4クロックが“High”という仕様になっている．

b. バリッド ペリフェラル アドレス（$\overline{\mathrm{VPA}}$）［入力］　この信号はセレクトされたデバイスが6800周辺 LSI であることをプロセッサに知らせるための信号である．プロセッサはこの信号を受けるとデータの転送をイネーブル信号（E）に同期して行い，また，割込みをオートベクタ割込み（9章参照）で行うようになる．

c. バリッド メモリ アドレス ($\overline{\mathrm{VMA}}$) [出力]　この出力信号は，アドレスバス上に有効なアドレスが出力されていることを示す．同時に，$\overline{\mathrm{VPA}}$ 入力に対する応答であること，すなわち，データ転送がイネーブルに同期して行われることを示す．

表 3.2 ファンクション コードとサイクル タイプ

FC2	FC1	FC0	Cycle Type
Low	Low	Low	(Undefined Reserved)
Low	Low	High	User Data
Low	High	Low	User Program
Low	High	High	(Undefined Reserved)
High	Low	Low	(Undefined Reserved)
High	Low	High	Supervisor Data
High	High	Low	Supervisor Program
High	High	High	Interrupt Acknowledge

表 3.3 信 号 の ま と め

信　号　名	ニモニック	入力/出力	アクティブ状態	スリーステート
アドレスバス	A1〜A23	出 力	High	○
データバス	D0〜D16	入出力	High	○
アドレスストローブ	$\overline{\mathrm{AS}}$	出 力	Low	○
リード/ライト	R/$\overline{\mathrm{W}}$	出 力	Read: High Write: Low	○
上位データストローブ, 下位データストローブ	$\overline{\mathrm{UDS}}$, $\overline{\mathrm{LDS}}$	出 力	Low	○
データ転送アクノレッジ	$\overline{\mathrm{DTACK}}$	入 力	Low	—
バス要求	$\overline{\mathrm{BR}}$	入 力	Low	—
バスグラント	$\overline{\mathrm{BG}}$	出 力	Low	—
バスグラントアクノレッジ	$\overline{\mathrm{BGACK}}$	入 力	Low	—
割込み優先レベル	$\overline{\mathrm{IPL0}}$, $\overline{\mathrm{IPL1}}$, $\overline{\mathrm{IPL2}}$	入 力	Low	—
バスエラー	$\overline{\mathrm{BERR}}$	入 力	Low	—
リセット	$\overline{\mathrm{RES}}$	入出力	Low	オープンドレイン
ホールト	$\overline{\mathrm{HALT}}$	入出力	Low	オープンドレイン
イネーブル	E	出 力	High	—
バリッド メモリ アドレス	$\overline{\mathrm{VMA}}$	出 力	Low	○
バリッド ペリフェラル アドレス	$\overline{\mathrm{VPA}}$	入 力	Low	—
ファンクション コード	$\overline{\mathrm{FC0}}$, $\overline{\mathrm{FC1}}$, $\overline{\mathrm{FC2}}$	出 力	High	○
クロック	CLK	入 力	High	—
電　源 (5V)	Vcc	入 力	—	—
接　地 (0V)	Vss	入 力	—	—

注) ○印はスリーステート信号であることを示す．

3.1.7 プロセッサ ステータス

ファンクション コード（FC0, FC1, FC2）　このファンクション コード出力は，表3.2に示すようにプロセッサのプログラム実行状態ならびにメモリ等のアクセスの種類を示す．すなわち，ユーザ モードかスーパバイザ モードか，プログラム フェッチ サイクルかデータ アクセス サイクルかが示される．さらに，これら3つの出力がすべて“High”レベルの場合はプロセッサが割込みを受け付けたこと（割込みアクノレッジ）を表す．

3.1.8 クロック（CLK）

クロックはプロセッサを動作させるための信号で，その最大周波数が 4, 6, 8, 10, 12 MHz の MPU が用意されている．

3.1.9 信号のまとめ

表3.3に今まで述べてきた信号のまとめを示す．この表には，信号名とそのニモニック，その信号の入出力の区別，アクティブ状態が“High”レベルなのか“Low”レベルなのかが記されている．さらに，この表の右端には，その信号がスリーステートになり得るかどうかが示されている．

3.2 バス オペレーション

この節では68000のバス オペレーションについて説明する．バス オペレーションにはデータ転送，バス アービトレーション，バス エラー，およびホールト，リセット等のオペレーションがある．なお，割込み発生時のバス オペレーション，6800周辺 LSI とのデータ転送オペレーション，また，DMA 時のバス オペレーションについては8,9および10章にて説明する．

3.2.1 データ転送オペレーション

プロセッサと外部デバイスとの間で行うデータ転送はデータ転送オペレーションに従って行われる．このオペレーションでは，データ バス（D0～D15），アドレス バス（A1～A23）および非同期データ バス制御信号（$\overline{\text{AS}}$, $\overline{\text{UDS}}$, $\overline{\text{LDS}}$, R/$\overline{\text{W}}$, $\overline{\text{DTACK}}$）が用いられる．また，そのとき実行されているバス オペレーションのタイプを示すためファンクシ

ョンコード (FC0~FC2) が出力される.

データ転送オペレーションには, リードサイクル, ライトサイクルおよびリードモディファイライトサイクルの3つの基本オペレーションがある. 以下, これら3つの基本オペレーションについて説明する.

a. リードサイクル　リードサイクルはプロセッサが外部デバイスからデータを読み取るオペレーションである. プロセッサが実行すべき命令は外部メモリに格納されているので, 命令フェッチもこのリードサイクルで行われる. 実行中のリードサイクルが命令であるか, データであるかはファンクションコード (FC0~FC1) に出力表示される.

図3.2にリードサイクルのタイミングチャートを示す. ここでリードサイクルとしてワードリード, 上位バイトリード, 下位バイトリードの3つが示されているが, その違いは表3.1でも示したように, 上位データストローブ ($\overline{UDS}$), 下位データストローブ ($\overline{LDS}$) のアサートの仕方が違うだけである, ここではワードリードサイクルについてそのオペレーションを説明する.

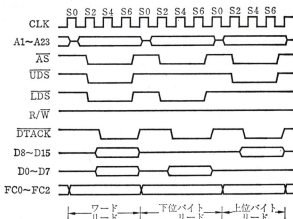



図 3.2 リードサイクルタイミング

S0状態では, まず, R/$\overline{W}$ ラインがリード状態 (High レベル) にされ, ファンクションコードが FC0~FC2 へ送出される. S0状態ではアドレスバスはまだハイインピーダンス状態である. 次に S1状態† になるとアドレスバスのハイインピーダンス状態が解除され, そこからアクセスすべきデバイスのアドレスが送出される. S2状態ではアドレス

† S1状態: S0とS2の間の CLK=Low レベルの状態である. 本書ではクロックサイクルの奇数状態を省略している場合がある.

ストローブ（$\overline{\mathrm{AS}}$）と上位データ ストローブ（$\overline{\mathrm{UDS}}$），下位データ ストローブ（$\overline{\mathrm{LDS}}$）がアサートされる．

$\overline{\mathrm{AS}}$ 信号を受けると外部デバイスはアドレスをデコードして，どのデバイスが選択されているかを知る．選択されたデバイスはデータ バス上にデータを送出し，また，同時にデータ送出を行ったことを示すデータ転送アクノレッジ（$\overline{\mathrm{DTACK}}$）をアサートする．

一方プロセッサは，S4 状態の終りで $\overline{\mathrm{DTACK}}$ がアサートされているかどうかチェックし，アサートされていればすぐに S5 状態に入る．もしこのときアサートされていなければ，S5 の状態に入らず Sw 状態（ウエート状態）に入る．この Sw 状態は図 3.3 に示すように，$\overline{\mathrm{DTACK}}$ がアサートされるまで挿入されるので，データ アクセス時間の遅いデバイスはその遅れ時間を含めて $\overline{\mathrm{DTACK}}$ をアサートすればよい．

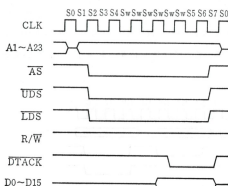



図 3.3　アクセスが遅いデバイスに対するリード サイクル タイミング

プロセッサは $\overline{\mathrm{DTACK}}$ がアサートされてデータ バス上にデータが確定していることを知ると，S6 状態の終りでデータを内部にラッチする．最後に S7 状態で，$\overline{\mathrm{UDS}}$, $\overline{\mathrm{LDS}}$ をネゲートし，$\overline{\mathrm{AS}}$ をネゲートしてリード サイクルを終了する．なおアドレス バスは，S7 状態が完了するまで有効である．

リード サイクルが終了すると，外部デバイスもデータ送出を停止するとともに，$\overline{\mathrm{DTACK}}$ をネゲートする．以上，リード サイクルのオペレーションをまとめてフローチャートにすると図 3.4 のようになる．

b. ライト サイクル　ライト サイクルはプロセッサが外部デバイスに対してデータを書き込むオペレーションである．図 3.5 はライト サイクルのタイミング チャートである．ライト サイクルにも，ワード ライト，上位バイト ライト，下位バイト ライトのサイクルがあるが，その違いはリード サイクルの場合と全く同じである．以下ワード ライト サイクルのオペレーションの説明を行う．

まず，S0 状態で，プロセッサはファンクション コードを FC0～FC1 へ出力する．このときアドレス バスはまだハイインピーダンス状態にある．S1 状態になるとアドレス バスのハイインピーダンス状態が解除され，そこからアクセスすべきデバイスのアドレスが

プロセッサ　　　　　　　　　　　　　　　　　　外部デバイス

デバイスをアドレス

S0 { 1) R/$\overline{\mathrm{W}}$ をリードにする
　　 2) ファンクションコードを FC0〜FC2 に送出する

S1 { 3) アドレスを A1〜A23 に送出する

S2 { 4) アドレスストローブ ($\overline{\mathrm{AS}}$) をアサートする
　　 5) 上位データストローブ ($\overline{\mathrm{UDS}}$) および
　　 下位データストローブ ($\overline{\mathrm{LDS}}$) をアサートする

データ読出し

1) アドレスをデコードする
2) データを D0〜D15 に送出する
3) データ転送アクノレッジ ($\overline{\mathrm{DTACK}}$) をアサートする

S4 { * $\overline{\mathrm{DTACK}}$ をチェックする ←…………

データ取込み

S6 { 1) データをラッチする

S7 { 2) $\overline{\mathrm{UDS}}$ と $\overline{\mathrm{LDS}}$ をネゲートする
　　 3) $\overline{\mathrm{AS}}$ をネゲートする

サイクル終了

1) データの D0〜D15 への送出を停止する
2) $\overline{\mathrm{DTACK}}$ をネゲートする

次のサイクルを開始

図 3.4 ワードリードサイクルフローチャート

S0 S2 S4 S6 S0 S2 S4 S6 S0 S2 S4 S6

CLK

A1〜A23

$\overline{\mathrm{AS}}$

$\overline{\mathrm{UDS}}$

$\overline{\mathrm{LDS}}$

R/$\overline{\mathrm{W}}$

$\overline{\mathrm{DTACK}}$

D8〜D15

D0〜D7

FC0〜FC2

ワードライト　下位バイトライト　上位バイトライト

図 3.5 ライトサイクルタイミング

送出される．次の S2 状態ではアドレスストローブ ($\overline{\text{AS}}$) がアサートされ，R/$\overline{\text{W}}$ が "Low" レベルになってライトサイクルであることを示す．S3 状態になるとデータバスからデータが送出され，また，S4 状態で上位データストローブ ($\overline{\text{UDS}}$)，下位データストローブ ($\overline{\text{LDS}}$) がアサートされる．

外部デバイス側では送出されたアドレスをデコードして，どのデバイスが選択されたかを知る．選択されたデバイスはデータバス上のデータを $\overline{\text{UDS}}$ または $\overline{\text{LDS}}$ を用いてメモリないしレジスタへ取り込み，プロセッサに対して $\overline{\text{DTACK}}$ をアサートする．なお，プロセッサでは $\overline{\text{DTACK}}$ がアサートされているかどうかをリードサイクルと同じように S4 の終りでチェックする．このとき $\overline{\text{DTACK}}$ がアサートされていなければ Sw 状態が S5 の前に挿入されるので，書込みに時間のかかるデバイスでは $\overline{\text{DTACK}}$ のアサートを遅らせればよい．

プロセッサは S4 または Sw 状態で $\overline{\text{DTACK}}$ がアサートされると，S5 の状態へ入り S6, S7 の状態へと進む．S7 状態では $\overline{\text{UDS}}$, $\overline{\text{LDS}}$ をネゲートし，さらに，$\overline{\text{AS}}$ をネゲートする．データバス上のデータおよび R/$\overline{\text{W}}$ のライト状態は S7 状態の間は保持され，次のサイクルの S0 状態になって始めて，データバス上からデータが除去され，R/$\overline{\text{W}}$ がリード状態に戻る．

また，デバイス側では $\overline{\text{AS}}$ がネゲートされたのに応じて $\overline{\text{DTACK}}$ をネゲートする．

以上のライトサイクルのオペレーションをまとめてフローチャートにすると，図3.6のようになる．

c. リードモディファイライトサイクル　リードモディファイライトサイクルは，プロセッサが外部のデバイスからデータを読み取り，それに論理演算操作を加えて変更したデータを同じアドレスのデバイスに書き込むオペレーションである．このサイクルはリードとライトオペレーションが連続して実行され，分割できないバスサイクルである．これはリードモディファイライトサイクルの重要な特徴である．リードモディファイライトサイクルはマルチプロセッサシステムにおけるプロセッサ間の交信で非常に有用な役割を果す．すなわち，2つ以上のプロセッサ間で共有されるデバイスまたはメモリを1つのプロセッサがリードモディファイライトサイクルで使用している間は，他のプロセッサはこのデバイスまたはメモリをアクセスすることができない．したがって，プロセッサ間の交信のために必要なフラグ等に対するリード/ライトの競合を避けることができる．

68000 では TAS (Test and Set) 命令だけがリードモディファイライトサイクルを使用する．TAS 命令はバイト操作しかできないので，ワードリードモディファイライトサ

プロセッサ　　　　　　　　　　　　外部デバイス

デバイスをアドレス

S0{ 1) ファンクションコードを FC 0〜FC 2 に送出する
S1{ 2) アドレスを A 1〜A 23 に送出する
S2{ 3) アドレスストローブ ($\overline{AS}$) をアサートする
　 4) R/$\overline{W}$ をライト状態にする
S3{ 5) データを D 0〜D 15 へ送出する
S4{ 6) 上位データストローブ($\overline{UDS}$) または
　 下位データストローブ ($\overline{LDS}$) をアサートする

データ書込み
1) アドレスをデコードする
2) データ D 0〜D 15 をストアする
3) データ転送アクノレッジ ($\overline{DTACK}$) をアサートする

S4{ * DTACK をチェックする ←…………

出力転送を終結
S7{ 1) $\overline{UDS}$, $\overline{LDS}$ をネゲートする
　 2) $\overline{AS}$ をネゲートする

サイクル終了
1) $\overline{DTACK}$ をネゲートする

S0{ 3) データをデータバス上から除去する
　 4) R/$\overline{W}$ をリード状態にする

次のサイクルを開始

図 3.6 ワードライトサイクルフローチャート

S0 S2 S4 S6 S8 S10 S12 S14 S16 S18
CLK
A1〜A23
$\overline{AS}$
$\overline{UDS}$ or $\overline{LDS}$
R/$\overline{W}$
$\overline{DTACK}$
D0〜D7 or D0〜D15
FC0〜FC2

図 3.7 バイトリードモディファイライトサイクルタイミング

プロセッサ　　　　　　　　　　　　　外部デバイス

デバイスをアドレス

S0{ 1) R/$\overline{\text{W}}$ をリードにする
　 2) ファンクションコードを FC0〜FC2 に送出する

S1{ 3) アドレスを A1〜A23 に送出する

S2{ 4) アドレスストローブ ($\overline{\text{AS}}$) をアサートする
　 5) 上位データストローブ ($\overline{\text{UDS}}$) または
　　下位データストローブ ($\overline{\text{LDS}}$) をアサートする

↓

データ読出し

1) アドレスをデコードする
2) データを D8〜D15 または D0〜D7 に送出する
3) データ転送アクノレッジ ($\overline{\text{DTACK}}$) をアサートする

S4{ * $\overline{\text{DTACK}}$ をチェックする

↓

データ取込み

S6{ 1) データをラッチする

S7{ 2) $\overline{\text{UDS}}$ または $\overline{\text{LDS}}$ をネゲートする
　 3) $\overline{\text{AS}}$ をネゲートする

↓

読出し終了

1) D8〜D15 または D0〜D7 からデータを除去する
2) $\overline{\text{DTACK}}$ をネゲートする

S8〜S13{ * データを修正する

↓

出力転送を開始

S14{ 1) R/$\overline{\text{W}}$ をライトにする
S15{ 2) データを D8〜D15 または D8〜D7 に送出する
S16{ 3) $\overline{\text{UDS}}$ または $\overline{\text{LDS}}$ をアサートする

↓

データ書込み

1) D8〜D15 または D0〜D7 上のデータをストアする
2) $\overline{\text{DTACK}}$ をアサートする

↓

出力転送を終結

S19{ 1) $\overline{\text{UDS}}$ または $\overline{\text{LDS}}$ をネゲートする
　　2) $\overline{\text{AS}}$ をネゲートする

↓

サイクル終了

1) $\overline{\text{DTACK}}$ をネゲートする

↓

S0{ 3) D8〜D15 または D0〜D7 上からデータを除去する
　 4) R/$\overline{\text{W}}$ をリード状態にする

↓

次のサイクルを開始

図 3.8 バイトリードモディファイライトサイクルフローチャート

イクルは存在しない．バイトリードモディファイライトサイクルのタイミングチャートを図3.7に，また，フローチャートを図3.8に示す．リードモディファイライトサイクルはリードサイクルとライトサイクルが続けて実行されるだけなので，詳細な動作の説明はここでは省略する．ただし，アドレスストローブ（$\overline{\mathrm{AS}}$）はこのサイクルの間ずっとアサートされるので注意されたい．

3.2.2 バスアービトレーション（Bus Arbitration）

バスアービトレーションは外部デバイスたとえばDMAC†がプロセッサから独立してデータバスを使用するときに必要となるオペレーションである．すなわち，バス制御能力を有するデバイス（プロセッサを含む）が2つ以上で1つのバスを共有すると，バスアクセス競合の問題が必ず生ずる．この競合を避けるためには，バス制御の支配権をどのデバイスがもつか，すなわち，どのデバイスがバスマスタになるかを決める必要がある．このオペレーションをバスアービトレーションという．

68000ではバスアービトレーションを可能にするために，バス要求（$\overline{\mathrm{BR}}$），バスグラント（$\overline{\mathrm{BG}}$），バスグラントアクノレッジ（$\overline{\mathrm{BGACK}}$）という3本の制御信号が用意されている．アービトレーション回路はプロセッサの外に，デイジチェインまたはプライオリティエンコーダ等を用いた制御回路として構成される．この場合，バスマスタになり得る外部デバイスはいくつあってもよいが，プロセッサは外部デバイスよりもバスマスタ権は低く格付けされることになる．

図3.9はバスアービトレーションのタイミングチャートである．以下この図に従いながら，バスアービトレーションのオペレーションを説明する．

バス制御能力を有するデバイスがバスを使用したいときには，まず，プロセッサに対してバス要求（$\overline{\mathrm{BR}}$）をアサートする．このようなデバイスが複数個あるときは，ワイヤドオアして$\overline{\mathrm{BR}}$へ入力する．プロセッサのオペレーションとこの信号は全く非同期であり，プロセッサはクロック（CLK）でこの信号をサンプリングし，同期化する．したがって，プロセッサが$\overline{\mathrm{BR}}$のアサートを内部的に認知するには$\overline{\mathrm{BR}}$入力後約0.5～1.5クロックサイクルの時間が必要である．

プロセッサは$\overline{\mathrm{BR}}$がアサートされたことを内部で認知すると，プロセッサがS0状態でない場合は次のクロックサイクルで，また，S0状態の場合は次の次のクロックサイクルでバスグラント（$\overline{\mathrm{BG}}$）をアサートする．すなわち，$\overline{\mathrm{BG}}$はプロセッサの実行状態のいか

† DMAC：Direct Memory Access Controller（10章参照）

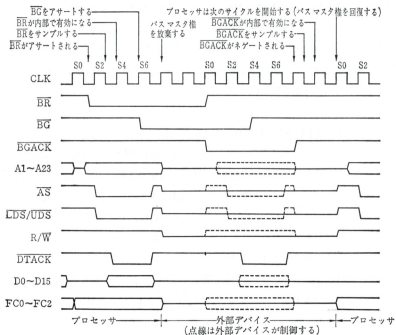



図 3.9　バス アービトレーション タイミング

んにかかわらず，$\overline{\mathrm{BR}}$ に対して約1.5〜2.5 クロック サイクル後に必ず応答する信号である．このため，プロセッサのバス マスタ権は外部デバイスより低く格付けられることになる．ただし，プロセッサはバス サイクルの途中であっても $\overline{\mathrm{BG}}$ のアサートを行うが，そのバス サイクルは最後まで正しく実行される．プロセッサはサイクルを終了し次第，バス マスタ権の放棄を行う．すなわち，プロセッサはアドレス バス，データ バス，$\overline{\mathrm{AS}}$, $\overline{\mathrm{LDS}}$, $\overline{\mathrm{UDS}}$, R/$\overline{\mathrm{W}}$ およびファンクション コードをハイ インピーダンス状態にする．

バス マスタ権になり得る外部デバイスが2つ以上存在すると，$\overline{\mathrm{BG}}$ は外部のバス アービトレーション回路を通して，バス要求を出しているデバイスに受け取られる．その外部デバイスは $\overline{\mathrm{BG}}$ を受け取ると，そのとき実行されているバス サイクルが終了するのを待ってバス マスタ権を獲得する．ここで，実行中のバス サイクルとはプロセッサもしくは他の外部デバイスがバス マスタになっているバス サイクルのことである．プロセッサのバス サイクルの終了は，アドレス ストローブ（$\overline{\mathrm{AS}}$）および データ転送アクノレッジ（$\overline{\mathrm{DTACK}}$）がネゲートされることによって知ることができ，他の外部デバイスが使用するバス

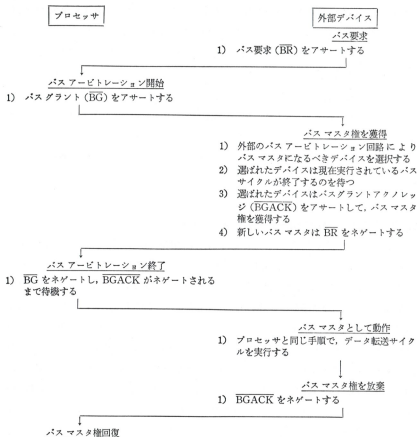



図 3.10 バス アービトレーション フローチャート

サイクルは $\overline{BGACK}$ および $\overline{DTACK}$ がネゲートされることによって知ることができる．

以上のようにして，$\overline{BR}$ をアサートしたデバイスがバス マスタ権を獲得すると，そのことをプロセッサに知らせるためにバス グラント アクノレッジ ($\overline{BGACK}$) をアサートするとともに，$\overline{BR}$ をネゲートする．プロセッサは $\overline{BR}$ がネゲートされたのを受けて $\overline{BG}$ ををネゲートしバス アービトレーションを終了する．プロセッサはその後は $\overline{BGACK}$ がネゲートされることを待つだけである．

新しいバス マスタは，アドレス バス，データ バスを $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, R/$\overline{W}$ およびファンクション コードで適当に制御することにより，データ転送のバス サイクルを形成することができる．図3.9では新しいバス マスタが形成するバス サイクルは1バス サイク

ルしか示していないが，2つ以上であってもよい．しかし，このときバスマスタは必ず $\overline{\mathrm{BGACK}}$ をアサートし続けておかねばならない．$\overline{\mathrm{BGACK}}$ は所定のバスサイクルが完全に終了してからネゲートする．これは，外部デバイスのバスマスタ権の放棄を意味する．

プロセッサは $\overline{\mathrm{BGACK}}$ がネゲートされたことを知ると，他の外部デバイスからのバス要求（$\overline{\mathrm{BR}}$）が受け付けられていないかぎり，バスマスタ権を回復する．もし，ここで他のデバイスが $\overline{\mathrm{BR}}$ をアサートしていれば，外部のアービトレーション制御回路で再アービトレーションを行い，外部デバイスが先にバスを占有することになる．

以上，バスアービトレーションのオペレーションをまとめると，図3.10のフローチャートのようになる．

3.2.3 バスエラーおよびホールトオペレーション

68000のデータ転送バスオペレーションは，$\overline{\mathrm{DTACK}}$ 信号を用いた非同期のハンドシェークデータ転送方式である．もし，何らかの原因で，たとえば，アドレスされたデバイスが存在しなかったり，電源が供給されていなかったりすると，ハンドシェークが成立し得ない．このような事態をここではバスエラーとよぶことにする．バスエラーの検出は外部回路にまかされているので，バスエラーとしてどういうものを設定するかはユーザの自由である．外部でバスエラーが検出されると，$\overline{\mathrm{BERR}}$ の入力信号を介してプロセッサにバスエラーの発生が知らされる．プロセッサとしてはバスエラーに対する処置は，$\overline{\mathrm{BERR}}$ 入力のほかに $\overline{\mathrm{HALT}}$ 信号がアサートされているかどうかで，再実行するかあるいは例外処理をするかが決められる．この項ではバスエラー時のバスオペレーションおよびホールト時のバスオペレーションについて説明する．

a. バスエラー例外処理　バスエラー例外処理が行われるのは，$\overline{\mathrm{HALT}}$ 信号がアサートされずに，$\overline{\mathrm{BERR}}$ だけがアサートされたときである．図3.11に示すようにプロセッサは，まず，$\overline{\mathrm{BERR}}$ がアサートされたことをS4またはSwのクロックの立下りで検出すると，データバスおよびアドレスバスをハイインピーダンス状態にする．プロセッサは次に $\overline{\mathrm{BERR}}$ がネゲートされたことを知って，例外処理シーケンスを開始する．

b. バスサイクルの再実行　プロセッサがバスサイクルを実行中に $\overline{\mathrm{BERR}}$ および $\overline{\mathrm{HALT}}$ がともにアサートされると，プロセッサはそのバスサイクルを再実行する．図3.12にバスサイクル再実行のタイミングチャートを示す．

プロセッサは，$\overline{\mathrm{BERR}}$ と $\overline{\mathrm{HALT}}$ が同時にアサートされていることをS4またはSwのクロックの立下りで検出すると，実行中のバスサイクルを終結し，データバスおよびア

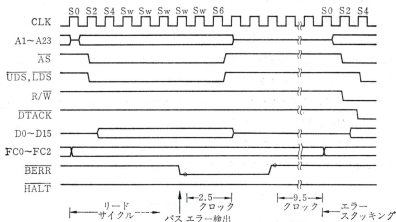



図 3.11 バス エラー タイミング

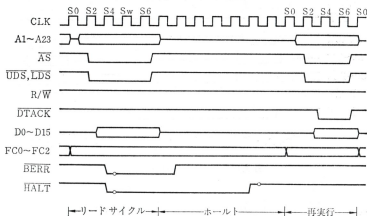



図 3.12 再実行バス サイクル タイミング

ドレス バスをハイインピーダンス状態にする．そして，プロセッサは $\overline{\text{HALT}}$ がネゲートされるまでホールト状態になる．次に $\overline{\text{HALT}}$ がネゲートされるとプロセッサは前と全く同じアドレス，データ，制御のもとにバス サイクルを再実行する．ここで，$\overline{\text{HALT}}$ のネゲートは必ず $\overline{\text{BERR}}$ をネゲートして1クロック後以降に行わねばならない．そうでないときは不正シーケンスとみなされ，ベクタ番号0の例外処理となるので注意が必要である．

プロセッサはリードモディファイ ライトサイクルに対してはバス サイクルの再実行を

行わない. これはリードモディファイ ライトサイクルは再実行しても結果が正しくなるとはかぎらないからである. リードモディファイ ライトサイクル実行中に再実行の要求があったときには, バス エラー例外処理となる.

c. ホールト オペレーション　$\overline{\text{BERR}}$ がネゲートのとき $\overline{\text{HALT}}$ だけがアサートされると, プロセッサはホールト状態になる. すなわち, プロセッサは実行中のバス サイクルが終了すると, アドレスバスおよびデータ バスをハイインピーダンス状態にして, $\overline{\text{HALT}}$ がネゲートされるまで次のバス サイクルの実行を停止する. 図3.13はホールト オペレーションを図示したものである. ここで, $\overline{\text{HALT}}$ がアサートされても $\overline{\text{DTACK}}$ がネゲートされたままであればバス サイクルは終了しないので, $\overline{\text{DTACK}}$ がアサートされ

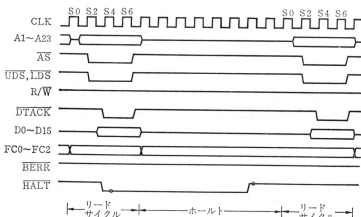



図 3.13　ホールト オペレーション

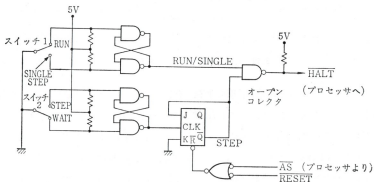



図 3.14　簡易形シングルステップ回路

ないかぎりホールト状態にならないことに注意が必要である．

ホールトオペレーションを用いると，シングルステップでのバスサイクルの実行が可能である．図3.14はシングルステップを実現する簡易な回路である．すなわち，スイッチ1をシングルステップ側へセットすれば，バスサイクルが1つ実行されるたびに $\overline{\mathrm{AS}}$ によってJKフリップフロップがリセットされるのでプロセッサはホールト状態になる．また，このときスイッチ2をウエートからステップ側へ1回倒すたびにJKフリップフロップはセットされるので，ホールト状態が解除されバスサイクルが1回だけ実行される．なお，この回路は，$\overline{\mathrm{HALT}}$ と $\overline{\mathrm{BERR}}$ や $\overline{\mathrm{RES}}$ の相互の影響については考慮していないので，実際に使うにあたっては注意が必要である．

プロセッサは $\overline{\mathrm{HALT}}$ 信号を受け付けていても，バスアービトレーションは通常どおり行われる．すなわち，$\overline{\mathrm{HALT}}$ はバスアービトレーションには何ら影響がない．バスアービトレーションは $\overline{\mathrm{AS}}$, $\overline{\mathrm{LDS}}$, $\overline{\mathrm{UDS}}$, R/$\overline{\mathrm{W}}$ の信号をバスから除去する，つまり，ハイインピーダンス状態にする機能である．

d. 2重バス障害 (Double Bus Faults)　バスエラー例外処理においてプロセッサはマシン状態に関する数ワードの情報をメモリ上に退避する．もし，バスエラーがこのデータ転送サイクルで発生すれば，バスエラーが2回続けて生じたことになる．これを2重バス障害という．2重バス障害になるとプロセッサはホールトする．一度バスエラー例外が発生したとき，次の命令を実行する前に発生するバスエラー例外はすべて2重バス障害となる．

再実行バスサイクルはバスエラー例外にも，2重バス障害にもならないことに注意すべきである．プロセッサは外部回路が再実行を要求するかぎり同じバスサイクルを何度でも繰り返す．

$\overline{\mathrm{BERR}}$ 信号はプロセッサが外部からリセット信号を受けたときのオペレーションにも影響を与える．プロセッサは $\overline{\mathrm{RES}}$ がアサートされるとプログラムの実行を開始すべく，リセット例外処理（8.3節参照）の実行によってメモリ上の情報を読み出す．もし，このときバスエラーが生じたら，これは2重バス障害とみなされ，プロセッサはホールトする．

2重バス障害のために生じたホールト状態は，外部からのリセットでなければホールト状態を解除することはできない．

3.2.4 リセットオペレーション

68000のリセット端子（$\overline{\mathrm{RES}}$）は双方向性信号である．これにより外部回路あるいはプ

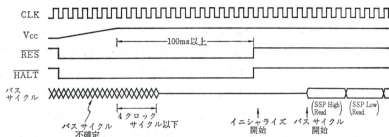



図 3.15 リセット オペレーション

ロセッサのいずれもがプロセッサを含む全体システムをリセットすることができる．図3.15はリセット オペレーションのタイミングチャートである．プロセッサを完全にリセットするためには，$\overline{\mathrm{RES}}$ と $\overline{\mathrm{HALT}}$ が外部から同時にアサートされなければならない．このアサートしている時間は，パワーオン時で100 ms，また，通常時で10 クロック サイクルの時間が必要である．

プロセッサはリセットを受け付けると，ベクタ番号0（0〜3番地）のアドレス情報を読み取りそれをスーパバイザ システム スタック（SSP）へロードする．次にベクタ番号1（4〜7番地）のアドレス情報を読み取りそれをプログラム カウンタへロードする．さらにプロセッサは，ステータス レジスタのＳビットをセット，Ｔビットをリセットし，割込みレベルを7にセットする．他のレジスタはリセット オペレーションによって影響されない．

プロセッサが **RESET** 命令を実行すると $\overline{\mathrm{RES}}$ 端子は124 クロック サイクルの間 “Low” レベルに駆動される．これは，プロセッサが外部回路をリセットする目的のものである．したがって，**RESET** 命令が終了すると，$\overline{\mathrm{RES}}$ 端子につながれている外部デバイスはすべてリセットされる．なお，**RESET** 命令の実行はプロセッサの内部状態やレジスタには全く影響を与えない．

4　命令の形式とアドレス形式

68000の大きな特徴のひとつは，豊富なアドレス形式を備えていることである．この章では，まず，この豊富なアドレス形式が機械語レベルでどのようにして実現されているかについて述べ，次に各アドレス形式の機能について述べる．

4.1 命 令 の 形 式

68000の命令は図4.1に示すような，4種類のワードすなわちオペレーションワード，イミディエイトオペランドワード，ソース実効アドレス拡張ワード，デスティネーション実効アドレス拡張ワードの組合せで構成される．オペレーションワードは命令の基本部分であり，命令の機能はこれによって決る．イミディエイトオペランドワードおよびソース/デスティネーション実効アドレス拡張ワードは各々1ワードないし2ワードで構成されるが，命令

15　　　　8 7　　　　0
オペレーションワード （1ワード）
イミディエートオペランドワード （0 or 1 or 2 ワード）
ソース実効アドレス拡張ワード （0 or 1 or 2 ワード）
デスティネーション実効アドレス拡張ワード （0 or 1 or 2 ワード）

図 4.1　命令の形式

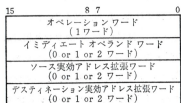
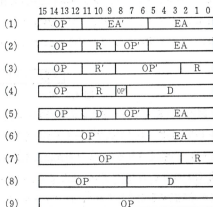
	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
(1)	OP ｜ EA′ ｜ EA
(2)	OP ｜ R ｜ OP′ ｜ EA
(3)	OP ｜ R′ ｜ OP′ ｜ R
(4)	OP ｜ R ｜ OP′ ｜ D
(5)	OP ｜ D ｜ OP′ ｜ EA
(6)	OP ｜ EA
(7)	OP ｜ R
(8)	OP ｜ D
(9)	OP

OP: オペレーションコード，EA: 実効アドレス，R: レジスタ No., D: データまたはディスプレースメント

図 4.2　オペレーションワードの形式

によっては省略される場合もある．また，イミディエイト オペランドとソース実効アドレス拡張ワードは同一命令のなかに同時には存在しない．したがって，68000の命令長は1ワードから5ワードまでとなる．

オペレーション ワードは命令の基本操作を指定する部分とオペランドを指定する部分により構成される．図4.2は68000の命令のオペレーション ワードの形式を示したものである．ここで，OP フィールドはオペレーションを指定する部分であり，EA，R およびD フィールドはオペランドを指定する部分である．したがって，(1)～(5) の形式の命令は2つのオペランドをもち，(6)～(8) の形式の命令は1つのオペランドをもっている．また，(9) のようにオペランドを全くもたない命令も存在する．また，(6)～(8) の命令形式の中には，オペレーション コードによってオペランドを暗黙的に指定しているものもある．

4.2 アドレス形式

オペランドがどこに存在するかを指定する形式をアドレス形式といい，各命令において オペランドが実際に存在する場所を示したものを実効アドレスという．

68000の命令では多くの場合，実効アドレスはオペレーション ワードの中の実効アドレス フィールドで指定される．実効アドレス フィールドは図4.3に示すようにアドレス形式フィールドとレジスタ フィールドに分けられ，これらの各フィールドがとる値によって表4.1に示すような12のアドレス形式に分けることができる．すなわち，68000の14個のアドレス形式のうち12の形式はこの実効アドレス フィールドで指定される．

アドレス形式	レジスタ

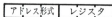



図 4.3　実効アドレス フィールドの構成

図4.2の R フィールドはレジスタの番号を表しているが，これは実効アドレス フィールドのうちアドレス形式フィールドがなくなったものと考えればよい．したがって，R フィールドで指定されるアドレス形式は表4.1の分類に含まれる．なお，図4.2における形式 (8) の命令は実効アドレスとしてディスプレースメント付プログラム カウンタ相対形式が暗黙的に仮定されており，D フィールドはそのディスプレースメントになっている．

表4.1に含まれない2つのアドレス形式は，クイック イミディエイト形式およびインプライド形式とよばれるものである．クイックイミディエイト形式とはオペレーションワードの中にイミディエイト データをもつものをいい，図4.2の形式 (4)，(5) でDフィ

表 4.1 実効アドレスフィールドによるアドレス形式

モードフィールド	レジスタフィールド	アドレス形式
0 0 0	Dn	データ レジスタ直接形式
0 0 1	An	アドレス レジスタ直接形式
0 1 0	An	アドレス レジスタ間接形式
0 1 1	An	ポスト インクリメント アドレス レジスタ間接形式
1 0 0	An	プリデクリメント アドレス レジスタ間接形式
1 0 1	An	ディスプレースメント付アドレス レジスタ間接形式
1 1 0	An	インデックス付アドレス レジスタ間接形式
1 1 1	0 0 0	短絶対アドレス形式
1 1 1	0 0 1	長絶対アドレス形式
1 1 1	0 1 0	ディスプレースメント付プログラム カウンタ相対形式
1 1 1	0 1 1	インデックス付プログラム カウンタ相対形式
1 1 1	1 0 0	イミディエイト形式

注) Dn: データ レジスタ番号, An: アドレス レジスタ番号

ールドがデータである場合がこれにあたる. また, インプライド形式とはオペレーション フィールドが暗黙的にオペランドを指定するものをいう.

以下, 表4.1のアドレス形式およびクイック イミディエイト形式, インプライド形式について主として MOVE 命令を例にとりながら説明する.

4.2.1 データ レジスタ直接形式

この形式ではレジスタ フィールドで指定される番号 n(=0~7) のデータ レジスタ Dn がオペランドとなる. したがって, 実効アドレス EA は

EA=Dn

となる.

[例] MOVE .W D1, D5

この命令は D1 のデータを D5 へ転送する命令である.

```
0 0 1 1      1 0 1    0 0 0       0 0 0    0 0 1
   ↑           ↑        ↑           ↑        ↑
MOVE Word   レジスタ  データ レジスタ         レジスタ
              D5     直接形式                  D1
       デスティネーション       ソース オペラ
       オペランド               ンド
```

4.2.2 アドレス レジスタ直接形式

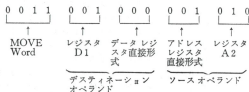
この形式では レジスタ フィールド で 指定 される 番号 n(=0~7) のアドレス レジスタ

An がオペランドとなる．実効アドレス EA は

EA=An

である．

［例］ MOVE .W A2, D1

この命令は A2 のデータを D1 へ転送する命令である．

4.2.3 アドレスレジスタ間接形式

この形式の実効アドレスは

EA=(An)

である．すなわち，アドレス レジスタ An の内容がアクセスされるメモリのアドレスを指し示している．

［例］ MOVE .W (A2), D3

この命令の動作は図 4.4 に示すとおりである．

4.2.4 ポストインクリメントアドレスレジスタ間接形式

この形式は，前項のアドレス レジスタ間接形式と実効アドレスの求め方は同じであるが，命令実行の最後にアドレス レジスタがインクリメントされるところが異なる．インクリメントされる値はその命令がバイト，ワード，または，ロング ワードのどの長さを操作する命令かによって，1，2 または 4 となる．すなわち，

EA=(An)；An + N ⟶ An (N=1,2 or 4)

ただし，アドレス レジスタがスタック ポインタ (A7) であるときは，バイト操作命令で

もインクリメント値は2となる．これはシステム スタック ポインタをワードの境界にしておくためである．

[例] MOVE .W (A2)+, D3

0 0 1 1	0 1 1	0 0 0	0 1 1	0 1 0
↑	↑	↑	↑	↑
MOVE Word	レジスタ D3	データ レジスタ 直接形式	ポスト インクリメント アドレス レジスタ 間接形式	レジスタ A2
	デスティネーション オペランド		ソース オペランド	

この命令の動作は図4.5に示すとおりである．

図 4.4 アドレス レジスタ間接形式の動作

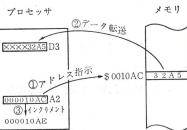



図 4.5 ポスト インクリメント アドレス レジスタ間接形式の命令の動作

4.2.5 プリデクリメント アドレス レジスタ間接形式

この形式ではまずアドレス レジスタの内容をデクリメントし，そのデクリメントされた値が実効アドレスを示す．すなわち，

An−N ⟶ An (N=1, 2 or 4)

EA=(An)

となる．なお，デクリメント値Nの決定方法は前項と同じく，命令が対象とするオペランドのデータ長がバイトかワードかロング ワードかにより，1, 2または4となる．さらに，アドレス レジスタがシステム スタック ポインタであれば，バイト操作命令であってもデクリメント値は2となる．

[例] MOVE .W −(A2), D3

0 0 1 1	0 1 1	0 0 0	1 0 0	0 1 0
↑	↑	↑	↑	↑
MOVE Word	レジスタ D3	データ レジスタ 直接形式	プリデクリメント アドレスレジスタ間接形式	レジスタ A2
デスティネーション オペランド			ソース オペランド	

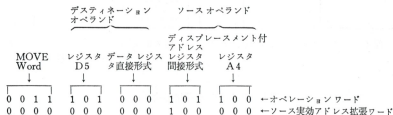
この命令の動作は図 4.6 に示すとおりである.

4.2.6 ディスプレースメント付アドレス レジスタ間接形式

このアドレス形式は 1 ワードの実効アドレス拡張ワードが必要であり，拡張ワードはこの場合ディスプレースメントとよばれる．オペランドのアドレスはアドレス レジスタの内容と 32 ビットに符号拡張されたディスプレースメントの和になる．すなわち，実効アドレスは

$$EA=(An)+d_{16}, \qquad d_{16}: 16 \text{ ビットディスプレースメントの値}$$

で与えられる．

［例1］ MOVE .W ＄20 (A4), D5

この命令の動作は図 4.7 に示すとおりである．

［例 2］ MOVE .W ＄20 (A4), ＄10 (A3)

この例はソース オペランドもデスティネーション オペランドもディスプレースメント付アドレス レジスタ間接形式である場合を示したものである．実効アドレスの拡張はソー

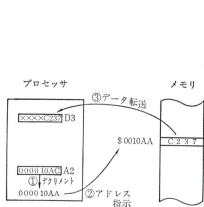



図 4.6 プリディクリメント アドレス レジスタ間接形式の命令の動作

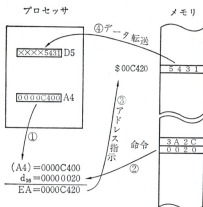



図 4.7 ディスプレースメント付アドレス レジスタ間接形式の命令の動作

ス，デスティネーション両方について前記のように各々行われる．

4.2.7 インデックス付アドレス レジスタ間接形式

このアドレス形式はインデックス レジスタを指定するために1ワードの実効アドレス拡張ワードが必要である．拡張ワードは図 4.8 の形式をしている．データ レジスタもアド

15	14　12	11	7	0
D/A	Rn	W/L	0 0 0	d_8

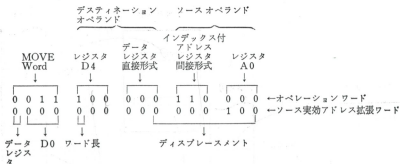
D/A：0のときデータ レジスタ，1のときアドレス レジスタ
Rn　：インデックス レジスタ No.
W/L：0のときインデックス レジスタの下位ワードが符号拡張される
　　　1のときインデックス レジスタ倍長ワードがそのままとられる
d_8　：8ビットのディスプレースメント

図 4.8 インデックス レジスタ指定のための拡張ワード

レス レジスタもインデックス レジスタになり得る．このアドレス形式の実効アドレスは次の式で与えられる．

$$EA=(An)+(Ri)+d_8$$

ここで，d_8 は8ビットのディスプレースメントで32ビットに符号拡張される．また Ri はアドレス レジスタ Ai，データ レジスタ Di のいずれかである（i=0〜7）．

［例］ MOVE .W　$04 (A0, D0), D4

この命令の動作は図4.9に示すとおりである.

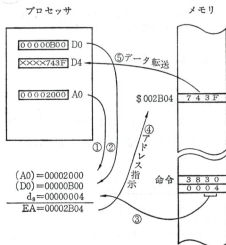



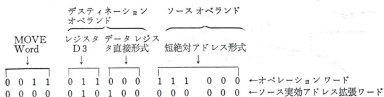
図 4.9　インデックス付アドレスレジスタ間接形式の命令の動作

4.2.8 短絶対アドレス形式

このアドレス形式では，1ワードの実効アドレス拡張ワードが必要である．実効アドレスとしてはこの拡張ワードの内容がとられる．すなわち，

EA＝Next Word

ここで，拡張ワードは16ビットであるので実効アドレスにするときには32ビットに符号拡張される．

［例］ MOVE .W　$ 0500, D3

この命令の動作は図 4.10 に示すとおりである.

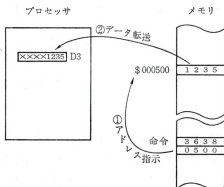



図 4.10 短絶対アドレス形式の命令の動作

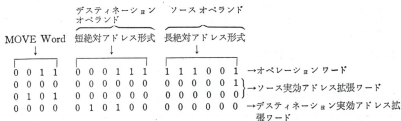
4.2.9 長絶対アドレス形式

このアドレス形式は 2 ワードの実効アドレス拡張ワードが必要であり，実効アドレスとしてこの 2 ワードの内容がとられる．すなわち，

EA=Next Two Words

ここで，拡張ワードの第 1 ワードはアドレスの上位部分であり，第 2 ワードはアドレスの下位部分である.

[例] MOVE .W $015000, $0500

```
                 デスティネーション   ソース オペランド
                 オペランド
MOVE Word    短絶対アドレス形式   長絶対アドレス形式
   ↓               ↓                  ↓
0 0 1 1      0 0 0 1 1 1        1 1 1 0 0 1   →オペレーション ワード
0 0 0 0      0 0 0 0 0 0        0 0 0 0 0 1 }
0 1 0 1      0 0 0 0 0 0        0 0 0 0 0 0 } →ソース実効アドレス拡張ワード
0 0 0 0      0 1 0 1 0 0        0 0 0 0 0 0   →デスティネーション実効アドレス拡張ワード
```

この命令の動作は図 4.11 に示すとおりである.

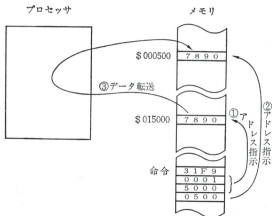



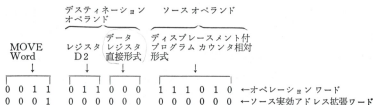
図 4.11　長絶対アドレス形式の命令の動作

4.2.10　ディスプレースメント付プログラム カウンタ相対形式

このアドレス形式は1ワードの実効アドレス拡張ワードが必要である．実効アドレスはプログラム カウンタの内容と拡張ワードであるディスプレースメントの和である．すなわち，

$$EA=(PC)+d_{16}$$

ここで，ディスプレースメント d_{16} は16ビットであるが，EA計算時には32ビットに符号拡張される．また，プログラム カウンタの内容は拡張ワードのアドレスを指している．

［例］ MOVE .W (LABEL), D3

この命令の動作は図4.12に示すとおりである．

4.2.11　インデックス付プログラム カウンタ相対形式

このアドレス形式は1ワードの実効アドレス拡張ワードが必要である．拡張ワードの形式は図4.8と同じであり，実効アドレスの計算式もアドレス レジスタがプログラム カウ

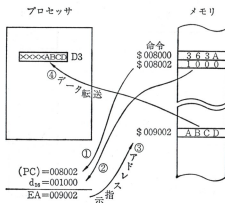



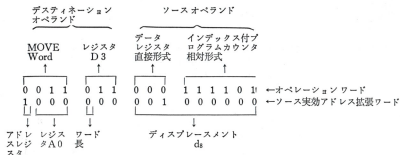
図 4.12 ディスプレースメント付プログラム カウンタ相対形式の命令の動作

ンタに変るだけである．すなわち，

$$EA=(PC)+(Ri)+d_8$$

ここで，ディスプレースメント d_8 は 8 ビットであるが 32 ビットに符号拡張される．また，プログラム カウンタは拡張ワードのアドレスを指している．

［例］ MOVE .W (LABEL) (A0), D3

```
デスティネーション                      ソースオペランド
オペランド
MOVE      レジスタ      データ       インデックス付プ
Word       D3          レジスタ      ログラムカウンタ
                       直接形式      相対形式
 ↑          ↑            ↑              ↑
0 0 1 1    0 1 1       0 0 0       1 1 1 1 0 1   ←オペレーションワード
1 0 0 0    0 0 0       0 0 1       0 0 0 0 0 0   ←ソース実効アドレス拡張ワード
↓  ↓        ↓                  ↓
アドレ レジス  ワード            ディスプレースメント
スレジ タA0   長                    d8
スタ
```

この命令の動作は図 4.13 に示すとおりである．

4.2.12 イミディエイト形式

このアドレス形式は 1 ワードまたは 2 ワードの実効アドレス拡張ワードが必要である．この拡張ワードはイミディエイト オペランドとよばれ，これが実効アドレスとなる．すなわち，

$$EA=\text{Next Word or Next Two Words}$$

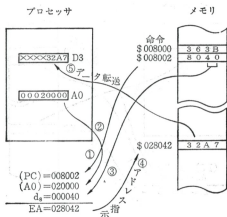



図 4.13　インデックス付プログラムカウンタ相対形式の命令の動作

ここで，イミディエイトオペランドは命令がバイトまたはワード操作命令のとき1ワードであり，ロングワード操作命令のとき2ワードである．

［例］ MOVE .W #$2345, $1000

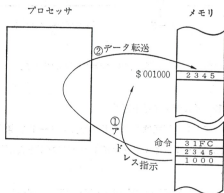



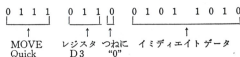
図 4.14　イミディエイト形式の命令の動作

この命令の動作は図4.14に示すとおりである．

4.2.13 クイック イミディエイト形式

クイック イミディエイト形式はイミディエイト データが拡張ワードに存在するのではなく，オペレーション ワードそのものに存在する．図4.2の命令の形式でいえば，(4) および (5) であり，このアドレス形式をもっている命令は3個である．すなわち，(4) の形式の命令として MOVEQ 命令，また，(5) の形式の命令として ADDQ 命令および SUBQ 命令があるだけである．しかも，イミディエイト データとして取り扱うことのできるデータの範囲も，(4) の形式で −128～127，また (5) の形式で1～8と非常に小さい．しかし，このアドレスの形式は命令ワード数を減らし，また，オペランドのリード サイクルも不要なため実行時間を短くするのに大きく貢献する．

［例］ MOVEQ #$5A, D3

```
0 1 1 1   0 1 1 0   0 1 0 1   1 0 1 0
└─────┘   └───┘└┘   └─────────────────┘
   ↑          ↑  ↑            ↑
 MOVE    レジスタ つねに  イミディエイト データ
 Quick      D3   "0"
```

この命令の動作は図4.15に示すとおりである．

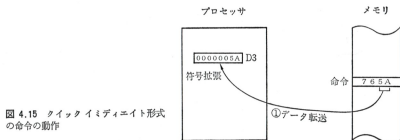



図 4.15 クイック イミディエイト形式の命令の動作

4.2.14 インプライド形式

インプライド形式とはオペレーション コードで暗黙的にオペランドが指定されるものをいう．たとえば，図4.2の (9) の形式の命令であっても，RTS (Return from Subroutine), RTE (Return from Exception), RTR (Return from Subroutine Restore CC) 命令では，オペランドとしてスタック ポインタやステータス レジスタまたはコンディション コードレジスタが暗黙的に指定されている．また，(6)，(7) の形式の命令である MOVE

to SR (Status Register), MOVE from SR, MOVE to CCR (Condition Code Register), MOVE to USP (User Stack Pointer), MOVE from USP の各命令については，ソースまたはデスティネーションの片方のオペランドは実効アドレスフィールドで指定するが，もう片方は暗黙的に指定されている形となっている．

5 命令の種類

この章では68000の命令セットを機能別に分類し，それぞれについて概説する．命令セットには，以下のオペレーションを実行する機能が含まれる．

(1) データ転送命令 (Data Movement)
(2) 算術演算命令 (Integer Arithmetic)
(3) 論理演算命令 (Logical)
(4) 10進演算命令 (Binary Coded Decimal)
(5) 桁移動命令 (Shifts and Rotates)
(6) ビット操作命令 (Bit Manipulation)
(7) プログラム制御命令 (Program Control)
(8) システム制御命令 (System Control)

命令によってはアドレス形式やステータス レジスタの状態と深いつながりをもつものもあるので，4章あるいは6章を適宜参照されたい．

5.1 データ転送命令

データ転送（MOVE）命令によって，基本的なデータ転送を行うことができる．この命令セットの中には一般のデータ転送命令以外に次に示すようないくつかの特殊なデータ転送命令がある．

MOVEM (Move Multiple Registers)
MOVEP (Move Peripheral)
EXG (Exchange Registers)
LEA (Load Effective Address)

表 5.1 データ転送命令の一覧表

命 令	サイズ	操 作	特 徴
MOVE	B W L	$(EA)_s \to EA_d$	ソース，デスティネーションに独立のアドレスモードが使える
MOVEA	W L	$(EA)_s \to An$	デスティネーション＝An の場合 W サイズの転送は符号拡張する
MOVEM	W L	$(EA)_s \to An$, Dn または An, Dn→EA	レジスタとメモリ間のブロック転送を行う
MOVEP	W L	$(EA)_s \to Dn$ または $Dn \to EA_d$	8ビット系ペリフェラルデータの転送に便利
MOVEQ	B	#×××→Dn	8ビットイミディエイトデータのセット
EXG	L	Rx↔Ry	レジスタ間のデータ交換用
SWAP	L	Dn(31：16)↔Dn(15：0)	同一レジスタ内の上位ワードと下位ワードのデータ交換用
LINK	—	An→SP@− SP→An SP+d→SP	サブルーチン等のデータエリアのネスティング用
UNLK	—	An→SP SP@+→An	（同上）
LEA	L	EA→An	実効アドレスを直接ロードする
PEA	L	EA→SP@−	

S: ソース, d: デスティネーション, B: バイト, W: ワード, L: ロングワード, EA: 実効アドレス
@ は関接アドレス形式を表す

PEA　　(Push Effective Address)

MOVEQ　　(Move Quick)

表 5.1 にデータ転送に関する命令の一覧表を示す．

a. 一般のデータ転送命令　MOVE 命令は 8 ビット（バイト），16 ビット（ワード），あるいは 32 ビット（ロングワード）のデータ転送を行う．ソースおよびデスティネーションオペランドをそれぞれ独立のアドレス形式で指定できる点が，この命令の大きな特徴になっている．ソースアドレスについてはすべてのアドレス形式が可能であり，デスティネーションアドレスについてはプログラムカウンタ修飾およびイミディエイトデータ形式を除くすべてのアドレス形式が使える．ただし，デスティネーションオペランドにアドレスレジスタを指定した場合，68000 はこれを MOVEA 命令と見なす．

68000 は周辺 I/O 装置のデータをメモリのデータと全く区別せずに扱う（メモリマップド I/O）ので特別の I/O 命令をもっていない．周辺 I/O 装置とのデータ転送を行う場合にも，周辺装置に固有のアドレスを割り当て，このアドレスを指定して MOVE 命令を実行する．転送するデータの長さやアドレス形式についてもメモリデータの場合と全く

同一である.

b. その他のデータ転送命令　データ転送命令の中には a. で述べた一般的な命令のほかにブロック転送を始めとする使用目的に応じた各種の命令が準備されている. 以下, これらの命令の特徴を説明する. 詳細は12章で説明する.

ブロック転送命令として MOVEM (Move Multiple Registers) 命令がある. この命令は内部レジスタ群とメモリとの間でのブロック転送を行う. この命令では転送したいレジスタを命令の第2ワードで指定でき, これによってデータ レジスタ8個, アドレス レジスタ8個のうちの任意のレジスタのブロック転送が行える. また転送語長もバイト, ワード, ロング ワードを指定できる.

MOVEP (Move Peripheral) 命令は特に8ビットのデータを送受する周辺 LSI と 68000 とのデータ転送に適した命令である. データ レジスタの2バイトまたは4バイトのデータと, 実効アドレスで指定されたロケーションの連続する2ないし4ワードの上位バイトまたは下位バイトとの間の転送を行う. 指定されたアドレスが偶数か奇数かによって上位バイトか下位バイトかが決る. この命令を使用すると16ビット バスに接続された8ビット系の周辺 LSI とのデータ転送が容易になる.

MOVEQ (Move Quick) 命令は, オペレーション ワードの8ビット フィールドに書かれているイミディエイト データをロング ワードに符号拡張してデータ レジスタに転送する命令である. この命令は1ワード長命令で4クロックで動作し, MOVE 命令に比べて高速処理される.

EXG (Exchange Registers) 命令は2つのレジスタ間の内容を交換する命令である. データ交換は32ビット (すなわちロング ワード) 単位で行い, データ レジスタ間, アドレス レジスタ間およびデータ レジスタとアドレス レジスタ相互で行える. さらに同一レジスタの上位ワードと下位ワードの交換には SWAP (Swap Register Halves) が準備されている.

実効アドレスのロード命令として LEA (Load Effective Address), PEA (Push Effective Address) 命令がある. LEA 命令は指定されたアドレス形式で実効アドレスを計算し, これをアドレス レジスタにロードする. また PEA 命令は実効アドレスをシステムスタックに格納する.

サブルーチンを多重にネスティングする場合の各サブルーチンで使用するスタックエリアの確保, 解放を行う命令として LINK (Link and Allocate) 命令と UNLK (Unlink) 命令がある. LINK 命令はアドレス レジスタ An の内容をシステム スタックに格納し,

アドレスを示すシステム スタック ポインタの内容をアドレス レジスタ An にロードする．次にサブルーチンの実行に必要なエリア分 d だけスタック ポインタを移動する．UNLK 命令は LINK 命令で確保したスタック領域を解除して 1 段浅いネスティング状態に戻る．LINK, UNLK 命令はサブルーチンの実行に必要なデータ エリアを確保しながらネスティングするので，リエントラントなサブルーチンを構成できる．LINK, UNLK 命令の詳細は 15 章で説明する．

5.2 算術演算命令

算術命令の中には，加算，減算，乗算，除算のほかに比較，テスト，クリア命令がある．表 5.2 に算術演算の一覧を示す．以下，機能の概要を示す．詳細は 13 章で説明する．

表 5.2 算術演算命令表

命令	サイズ	操作	特長
ADD/SUB	B W L	Dn±(EA)→Dn (EA)±Dn→EA	
ADDA/SUBA	W L	An+(EA)→An	ワード指定の場合，ソース オペランドをロングワードに符号拡張して演算する
ADDI/SUBI	B W L	(EA)±#×××→EA	B, W, L の任意サイズのイミディエイト データが扱える
ADDQ/SUBQ	B W L	(EA)±#×××→EA	1〜8 ビットイミ ディエイト データの処理を行う
ADDX/SUBX	B W L W L	Dx±Dy±X→Dx Ax@−±Ay@−±X→Ax@−	
MULS/MULU	W*W→L	Dn*(EA)→Dn	
DIVS/DIVU	L÷W→L	Dn/(EA)→Dn	
EXT	B→W W→L	Dn[7]→Dn[8−15] Dn[15]→Dn[16−31]	
NEG	B W L	0−(EA)→EA	
NEGX	B W L	0−(EA)−X→EA	
CMP	B W L	Dn−(EA)	
CMPA	W L	An−(EA)	
CMPI	B W L	(EA)−#×××	
CMPM	B W L	Ax@+−Ay@+	
TST	B W L	(EA)−0	
TAS	B	(EA)−0, 1→EA[7]	複数プロセッサの同期に有効

a. 加算・減算命令　ADD (Add Binary) 命令および SUB (Subtract Binary) 命令はデータ オペレーションとアドレス オペレーションの両方に使用できる．データ オペレ

ーションはデスティネーション オペランドにソース オペランドを加算あるいは減算し，その結果をデスティネーションのデータ レジスタまたはメモリに格納する．データ オペレーションではすべてのオペランド サイズが使用できる．アドレス オペレーションはデスティネーション オペランドとしてアドレス レジスタを指定する場合であり，ADDA (Add Address)，SUBA (Subtract Address) 命令によって実行される．アドレス オペレーションのオペランド サイズは16または32ビットに限定される．ワード指定の場合にはソースオペランドをワードからロング ワードに符号拡張して32ビット データとして演算される．

イミディエイト データの加算・減算命令として ADDI (Add Immediate)，ADDQ (Add Quick)，SUBI (Subtract Immediate)，SUBQ (Subtract Quick) 命令がある．ADDI, SUBI 命令はデスティネーション オペランドとイミディ エイト データを加算あるいは減算を行ってデスティネーション オペランドに結果を格納する．データのサイズはバイト，ワード，ロング ワードのすべてが指定できる．ADDQ, SUBQ 命令は1～8のイミディエイト データとデスティネーション オペランド間の演算を行う．この命令は ADDI, SUBI に比べて実行クロック サイクル数およびプログラム バイト数が少ないという特徴がある．SUB 命令の変形として0からデスティネーション オペランドを減算する NEG (Negate) 命令がある．

b. 乗算・除算命令　乗算命令 MULS(Multiply Signed)，MULU(Multiply Unsigned) 命令はワード長オペランドの乗算を行ってロング ワード長の積を得る．オペランドがレジスタの場合は下位16ビットのみが取り出され上位16ビットは無視される．除算命令 DIVS (Divide Signed)，DIVU (Divide Unsigned) はロング ワード長のデスティネーション オペランドをワード長のソース オペランドで除算し，ワード長の商と余りを得る．商はデスティネーションの下位ワードに，また余りは上位ワードに格納される．除数が0の場合はトラップが発生し，例外処理を開始する．また命令完了前にオーバフローが生じると V フラグがセットされ命令は実行されない．この場合デスティネーション オペランドは元の値が保持される．乗算・除算共符号付，および符号なしオペランドの演算ができる．符号付の場合は MULS, DIVS 命令を，また符号なしの場合は MULU, DIVU 命令を使用する．

c. 比較命令　比較命令には CMP(Compare)，アドレス レジスタの比較を行う CMPA (Compare Address)，イミディエイト データとの比較を行う CMPI(Compare Immediate) メモリ データ間の比較を行う CMPM (Compare Memory) および0との比較を行う TST

(Test) がある．テスト命令の変形として，TAS (Test and Set Operand) 命令がある．この命令はデスティネーション オペランドのテストを行うと同時に，そのオペランドの最上位ビットを "1" にセットする．テストとセットがひとつのバス サイクル中で行われるので，複数のプロセッサの同期等に有効である．

比較命令はデスティネーション オペランドからソース オペランドを減算し，その結果に従ってコンディション コードをセットする．このときオペランド自身は変化しない．

d. 拡張命令セット　拡張命令セットを使用することによって，倍精度演算および長さの異なるオペランド間の算術演算を実行することができる．これらの命令としては ADDX (Add Multi-precision)，SUBX (Subtract Multi-precision)，EXT (Sign Extend)，NEGX (Negate Multi-precision) がある．この命令はデスティネーション オペランドとソース オペランドとコンディション コードの中の X フラグの間の演算を行う．たとえば 64 ビット長のデータの加算を行う場合，下位 32 ビットの加算は ADD 命令で行い，上位 32 ビットの加算は ADDX 命令で行う．ADDX 命令を用いることで下位からの桁上りに対して正しく 64 ビット データの演算が行える．

e. クリア命令　CLR (Clear Operand) 命令はオペランドに 0 をセットする命令である．

5.3 論理演算命令

論理演算命令として AND (論理積)，OR (論理和)，EOR (排他的論理和)，NOT (否定) 命令がある．これらはすべてのオペランド サイズに対して使用できる．また，類似のイミディエイト命令セット (ANDI, ORI, EORI 命令) も，すべてのオペランド サイズに対して実行できる．表 5.3 に論理演算命令の一覧表を示す．命令の詳細説明は 13 章で行う．

5.4 2 進化 10 進数演算命令

2 進化 10 進数 (Binary Coded Decimal) の演算命令に ABCD (Add Digits)，SBCD (Subtract Digits)，NBCD (Negate Digits) がある．これらの命令は 2 進化 10 進形式で記憶されているデータ間の加算減算処理を行い，長いデータ間の処理が容易にできる．表 5.4 に 2 進化 10 進数の演算命令表を示す．各命令の詳細は 13 章で述べる．

表 5.3　論理演算命令表

命　令	サイズ	操　　作
AND	B W L	Dn∧(EA)→Dn
		(EA)∧Dn→EA
ANDI	B W L	(EA)∧#×××→EA
OR	B W L	Dn∨(EA)→Dn
		(EA)∨Dn→EA
ORI	B W L	(EA)∨#×××→EA
EOR	B W L	(EA)⊕Dy→EA
EORI	B W L	(EA)⊕#×××→EA
NOT	B W L	−(EA)→EA

表 5.4　2進化 10 進数演算命令表

命　令	サイズ	操　　作
ABCD	B	Dx_{10}† $+Dy_{10}+X\rightarrow Dx$
		$Ax@_{-10}+Ay@_{-10}+X\rightarrow Ax@-$
SBCD	B	$Dx_{10}-Dy_{10}-X\rightarrow Dx$
		$Ax@_{-10}-Ay@_{-10}-X\rightarrow Ax@-$
NBCD	B	0−(EA)−X→EA

† サフィックスの 10 は，データ (Dx, Dy) およびアドレス (Ax, Ay) が 2 進化 10 進形式であることを示している．

5.5 桁移動操作命令

シフトおよびローテート命令　算術シフト命令 ASL, ASR (Arithmetic Shift Left/Right), 論理シフト命令 LSL, LSR (Logical Shift Left/Right) によってシフト動作が行える．ローテート命令には拡張フラグを含まない ROL, ROR (Rotate without X Left/Right) および拡張フラグを含む ROXL, ROXR (Rotate through X Left/Right) がある．すべてのシフト操作とローテート操作はレジスタとメモリのどちらにおいても実行できる．レジスタのシフトとローテートではオペランドはすべてのサイズをとることができる．シフトあるいはローテートするビット数の指定法として 2 種類が準備されている．命令の中で直接カウント数を指定する場合は 1〜8 ビットの範囲となる．データレジスタで間接的に指定する場合は 0〜63 ビットの範囲が指定できる．表 5.5 にシフトおよびローテート命令の一覧を示す．各命令の詳細は 14 章で説明される．

表 5.5　算術・論理シフトおよびローテート命令表

命　令	サイズ	操　　作
ASL	B W L	
ASR	B W L	
LSL	B W L	
LSR	B W L	
ROL	B W L	
ROR	B W L	
ROXL	B W L	
ROXR	B W L	

5.6 ビット操作命令

ビット操作命令として BTST (Bit Test), BSET (Bit Test and Set), BCLR (Bit Test and Clear), BCHG (Bit Test and Change) 命令がある．これらの命令はデスティネーションオペランドの任意の1ビットに対してビット操作を行うもので，メモリ上のデータについては8ビット，データ レジスタについては32ビットのデータのうちの1ビットが指定できる．ビット位置の指定の方法にはデータ レジスタで行う方法と命令の第2ワード目で指定する方法がある．BTST 命令は指定ビットの状態をテストし，その結果をステータス レジスタのZビットに反映する．BSET, BCLR, BCHG 命令はテスト操作に引き続いて，テスト ビットに対してそれぞれ1，0のセットあるいは反転操作を行う．表5.6にビット操作命令の一覧を示す．各命令の詳細は14章で述べる．

表 5.6 ビット操作命令表

命 令	サイズ	操 作
BTST	B L	~bit of (EA)→Z
BSET	B L	~bit of (EA)→Z 1→bit of (EA)
BCLR	B L	~bit of (EA)→Z 0→bit of (EA)
BCHG	B L	~bit of (EA)→Z ~bit of (EA)→bit of EA

・Z= ステータス レジスタのビット 2 (Zero)
・テストするビット位置はデータ レジスタで指定する方法とリテラル値で指定する方法がある.
・~はビット反転を表す (1=0, 0=1).

5.7 プログラム制御命令

a. 条件付ブランチ命令 コンディション コードの状態を判定して，その結果によってプログラム カウンタ (PC) を操作する命令として Bcc (Branch Conditionally), DBcc (Decrement Counter and Branch until Condition True or Count＝−1) 命令がある．状態判定はコンディション コードの4つのフラグに対して16種類 (Bcc に対しては14種類) が用意されている．ブランチ アドレスはプログラム カウンタからの相対値として与えられ，-2^{15}～$2^{15}-1$ バイト離れた位置までブランチできる．DBcc 命令はデータ レジスタ Dn をカウンタとして使用し，命令実行毎に Dn の内容をデクリメントし Dn=0 になるまで分岐アドレスにブランチし，Dn＝−1 になると次の命令が実行される．Scc (Set Conditionally) はコンディション コードの状態を判定し，条件が真 (True) の場合，実効アドレスで指定されたバイト データのすべてのビットに “1” をセットし，条件が偽 (False) の場合は “0” をセットする．条件表を表 5.7 に示す．演算が符号付であるかどうかにより，算術的に同じ意味でも条件が違う．表 5.8 の条件式とコンディション (条件) コードの対応をみて，目的に合った条件式を使用すればよい．

b. 無条件ブランチまたはジャンプ命令 表 5.9 にプログラム操作命令の一覧を示す．

ブランチ命令として BRA (Branch Always), BSR (Branch to Subroutine) 命令があり，ジャンプ命令として JMP (Jump), JSR (Jump to Subroutine) 命令がある．BRA 命令の分岐アドレスはプログラム カウンタから相対アドレスとして 8 または 16 ビット ディ

表 5.7 条件分岐の判定条件

指定	判定条件	適用可能性		
		Bcc	DBcc	Scc
CC	キャリィクリア	○	○	○
CS	キャリィセット	○	○	○
EQ	等しい	○	○	○
F	真か	×	○	○
GE	>=	○	○	○
GT	>	○	○	○
HI	ハイ	○	○	○
LE	<=	○	○	○
LS	ローあるいは同じ	○	○	○
LT	<	○	○	○
MI	マイナス	○	○	○
NE	≠	○	○	○
PL	プラス	○	○	○
T	常に真	×	○	○
VC	オーバフローなし	○	○	○
VS	オーバフローあり	○	○	○

表 5.8 符号条件コードの対応

条件式	符号なし	符号あり
X<Y	HI	GT
X≦Y	CC	GE
X≧Y	LS	LE
X>Y	CS	LT

表 5.9 プログラム操作命令表

種類	命令	操作
条件付操作	Bcc	条件ブランチ（8または16ビットディスプレースメント）
	DBcc	条件テスト，Dn−1→Dn，If Dn=−1 then ブランチ
	Scc	条件=1: 1→EA 条件=0: 0→EA　（16ビットディスプレースメント）
無条件分岐	BRA	常にブランチ（8または16ビットディスプレースメント）
	BSR	サブルーチンブランチ（　〃　）
	JMP	ジャンプ
	JSR	サブルーチンジャンプ
リターン	RTS	サブルーチンからのリターン
	RTR	サブルーチンからのリターンおよびCC回復

スプレースメントで指定され，JMP 命令の分岐アドレスは種々提供されている．これについては 15 章で説明する．

JSR, RTS (Return from Subroutine) 命令はサブルーチンのネスティングを操作する．メイン プログラムに書かれた JSR 命令を実行するとプログラム カウンタをスタックへ退避しサブルーチンにジャンプする．サブルーチンでは最後に書かれた RTS 命令を実行するとスタックの先頭のデータをプログラム カウンタに回復し，メイン プログラムに戻る．

RTR (Return from Subroutine Restore CC) 命令はスタックからプログラム カウンタとコンディション コードを回復する命令である．

各命令の詳細は 15 章で説明される．

5.8　システム制御命令

表 5.10 にシステム制御に関係する命令をまとめて示す．これらはトラップ命令，ステータス レジスタ操作命令および特権命令に分類される．トラップ命令，ステータス レジスタ操作命令は 16 章，特権命令は 17 章で詳しく説明する．本節では，以下に各命令の概要を述べる．

表 5.10　システム制御命令表

種　類	命　令	操　　作
トラップ発生命令	TRAP	PC→SSP@−，SR→SSP@− (ベクトル)→PC
	TRAPV	If V=1 then TRAP
	CHK	If Dn<0 or Dn>(<EA>) then TRAP
ステータスレジスタ操作	MOVE EA to CCR	(EA)→CCR
	MOVE SR to EA	(SR)→EA
	ANDI to CCR	(CCR)∧#×××→CCR
	EORI to CCR	(CCR)⊕#×××→CCR
	ORI to CCR	(CCR)∨#×××→CCR
特権命令	MOVE EA to SR	(EA)→SR
	ORI to SR	(SR)∨#×××→SR
	ANDI to SR	(SR)∧#×××→SR
	EORI to SR	(SR)⊕#×××→SR
	MOVE to/from USP	(USP)→An
		(An)→USP
	RTE	SP@+SR；SP@→PC
	RESET	外部デバイスをリセット
	STOP	#×××→SR，ストップ

a. トラップ発生命令　トラップ発生命令はソフトウェア割込みを発生する命令でTRAP (Trap), TRAPV (Trap if Overflow Set), CHK (Check Register against Bounds) 命令がある．TRAP命令を実行するとプログラム カウンタとステータス レジスタをスタックへ退避して，命令で指定したベクトル番号に対応するアドレスがセットされる．TRAPV命令を実行するとコンディション コードのVフラグがテストされ，Vフラグが"1"の場合は例外処理を開始する．TRAPV 例外処理ベクトルはシステムで発生されるので命令で指定しなくてよい．Vフラグが"0"の場合は何もせずに直ちに次の命令に移る．CHK命令はデータ レジスタの下位16ビットについて判定し，0より小さいかまたはソース オペランドで指定した上限値より大きい場合は例外処理を開始する．

b. ステータス レジスタ操作命令　特権命令にならないステータス レジスタ操作命令に MOVE EA to CCR (Condition Code Register), MOVE SR (Status Register) to EA, ANDI to CCR, EORI to CCR, ORI to CCR 命令がある．

c. 特権命令　特権命令はスーパバイザ状態でのみ実行可能な命令で，システム制御命令のうち特にシステムの安全性に影響の大きな命令である．

ステータス レジスタを操作する特権命令として MOVE EA to SR (Move to Status Register), ORI to SR, AND to SR, EOR to SR 命令がある．いずれもステータス レジスタ16ビットの操作命令である．なおステータス レジスタの下位8ビットすなわちコンディション コード レジスタを指定した場合は特権命令にならない．

MOVE USP (Move User Stack Pointer) 命令はアドレス レジスタとユーザ システム スタックポインタ USP の間のデータの転送を行う．スーパバイザ モードでユーザ システム スタックを使う場合に必要になる．

RTE (Return from Exception) 命令は例外処理からのリターン命令で例外処理プログラムの最後に書かれる．この命令を実行するとスーパバイザ システム スタックからステータス レジスタとプログラム カウンタの内容が回復され，例外処理発生の前の状態に戻る．

RESET 命令の実行によってリセット信号が発生し，リセット ラインにつながるすべての外部装置をリセットする．また STOP (Load SR/Stop) 命令では16ビットのイミディエイト データをステータス レジスタに転送し，プログラム カウンタが次のアドレスを示した状態でプロセッサは命令のフェッチおよび実行を停止する．トレース，割込み，リセットの例外処理が生じた場合に命令の実行を再開する．STOP 命令を実行する際，ステータス レジスタのTビット (ビット15) が"1"の場合は直ちにトレース例外処理に移る．

以上説明した命令のほかに，いずれの分類にも入らない命令として NOP（No Operation）がある．この命令の命令長は1バイトであり何もせずに次の命令に移るもので，実行サイクル数やプログラム ワード数を調整する場合に有効である．

またオペレーション ワードの上位4ビットが“1010”または“1111”の命令は，未実装命令とよばれ，いわゆるエミュレーション用のものである．16章で詳しく説明する．

6　フラグと算術論理

この章では68000のフラグの種類と機能の説明を行う．図6.1に示すように68000には次の5個のフラグがある．

（1）N……ネガティブ フラグ（Negative）

（2）Z……ゼロ フラグ（Zero）

（3）V……オーバフロー フラグ（Overflow）

（4）C……キャリィ フラグ（Carry）

（5）X……エクステンド フラグ（Extend）

これらのフラグ ビットはまとめてコンディション コードとよばれ，コンディション コード レジスタ（CCR）に格納される．CCR はステータス レジスタの下位バイトに位置している（2章参照）．CCR へはスーパバイザ状態でもユーザ状態でもアクセスできる．

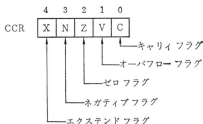



図 6.1　68000のコンディション コード レジスタ（CCR）とフラグ構成

各フラグは算術論理演算命令，シフト命令，ローテート命令およびビット操作命令などの実行結果を反映する．68000では演算のサイズとして1，8，16，32ビットがあるが，フラグの設定は，指定されたオペランド サイズ内の実行結果を反映する．

6.1　命令実行とフラグの設定

68000が命令を実行した場合，その実行結果によって各フラグは次のいずれかの状態と

表 6.1 フラグ機能に関する記号の定義

記 号	意 味
S_m	ソース オペランドの最上位ビット
D_m	デスティネーション オペランド最上位ビット
R_m	実行結果の最上位ビット
R_0	実行結果の最下位ビット
D_n	デスティネーション オペランドの第nビット
r	桁移動命令における桁移動の数

なる.

（1） 0に設定

（2） 1に設定

（3） 未定義（0, 1のいずれとなるか保証されない）

（4） 命令実行前の値を保持

各フラグの設定条件の説明に必要な記号を表6.1に定義しておく.

6.2 各フラグの説明

表6.2は，フラグに影響を与える命令とフラグの設定条件に関するものである．表6.2に現れない命令を実行してもフラグの値は変化しない．特に，アドレス レジスタを使う ADDA, SUBA, MOVEA 命令はフラグに影響しないことに注意を要する．以下，各フラグについて説明する.

6.2.1 N フ ラ グ

N (Negative) フラグは，命令の実行結果が負かどうかを示すフラグである．このフラグは，実行結果が負のとき1，正または0のとき0が設定される．一般的には，実行結果が負とは最上位ビット R_m が1であることに等しいため，次式が成り立つ.

$$N=R_m \qquad (6\text{-}1)$$

表6.2でも示すように大部分の命令では式（6-1）が成立するが，DIVS, DIVU 命令，2進化10進数演算命令，CHK 命令および CCR を操作する命令では式（6-1）は成り立たない.

DIVS, DIVU 命令では，商（結果の下位バイト）が負のときNフラグは1に設定される．商のオーバフローが発生した場合にはNフラグは未定義である．除数が0の場合Nフラ

表 6.2　フラグに影響を与える命令とフラグの設定条件

命令の種類	命　令	X	N	Z	V	C	特殊な条件
算術演算命令	ADD, ADDI, ADDQ	*	*	*	!	!	$V=S_m\cdot D_m\cdot\overline{R_m}+\overline{S_m}\cdot\overline{D_m}\cdot R_m$ †1 $C=S_m\cdot D_m+D_m\cdot\overline{R_m}+S_m\cdot\overline{R_m}$
	ADDX	*	*	!	!	!	V, C 同上 $Z=Z\cdot\overline{R_m}\cdot\cdots\cdot\overline{R_0}$
	CLR	—	0	1	0	0	
	CMP, CMPA, CMPI, CMPM	—	*	*	!	!	$V=S_m\cdot\overline{D_m}\cdot R_m+\overline{S_m}\cdot D_m\cdot\overline{R_m}$ $C=S_m\cdot\overline{D_m}+\overline{D_m}\cdot R_m+S_m\cdot R_m$
	DIVS, DIVU	—	*	*†2	!	0	V＝商のオーバフロー
	EXT	—	*	*	0	0	
	MULS, MULU	—	*	*	0	0	
	NEG	*	*	*	!	!	$V=D_m\cdot R_m$ $C=D_m+R_m$
	NEGX	*	*	!	!	!	V, C 同上 $Z=Z\cdot\overline{R_m}\cdot\cdots\cdot\overline{R_0}$
	SUB, SUBI, SUBQ	*	*	*	!	!	$V=S_m\cdot\overline{D_m}\cdot R_m+\overline{S_m}\cdot D_m\cdot\overline{R_m}$ $C=S_m\cdot\overline{D_m}+\overline{D_m}\cdot R_m+S_m\cdot R_m$
	SUBX	*	*	!	!	!	V, C 同上 $Z=Z\cdot\overline{R_m}\cdot\cdots\cdot\overline{R_0}$
	TAS, TST	—	*	*	0	0	
論理演算命令	AND, AND, EOR, EORI, NOT, OR, ORI	—	*	*	0	0	
2進化10進数演算命令	ABCD	*	U	!	U	!	C＝10進数のキャリィ $Z=Z\cdot\overline{R_m}\cdot\cdots\cdot\overline{R_0}$
	NBCD, SBCD	*	U	!	U	!	C＝10進数のボロー $Z=Z\cdot\overline{R_m}\cdot\cdots\cdot\overline{R_0}$
桁移動命令	ASL, ASR LSL, LSR　(r=0) ROL, ROR	—	*	*	0	0	
	ROXL, ROXR(r=0)	—	*	*	0	!	C=X
	ASL	*	*	*	!	!	$V=D_m\cdot(\overline{D_{m-1}}+\cdots+\overline{D_{m-r}})$ $+\overline{D_m}\cdot(D_{m-1}+\cdots+D_{m-r})$ $C=D_{m-r+1}$
	LSL, ROL, ROXL	*	*	*	0	!	$C=D_{m-r+1}$
	ASR,	*	*	*	0	!	$C=D_{r-1}$

命令の種類	命　令	X	N	Z	V	C	特殊な条件
	LSR, ROR, ROXR	*	*	*	0	!	
ビット操作命令	BCHG, BCLR, SET, BTST	—	—	!	—	—	$Z=\overline{Dn}$
その他の命令	MOVE, MOVEQ, SWAP	—	*	*	0	0	
	CHK	—	*†3	U	U	U	
	MOVE to CCR			!			ソース オペランドの値
	MOVE to SR			!			
	RTE			!			SSP の値
	RTR			!			USP の値
	STOP			!			イミディエイト値

—: 命令実行前の値を保持，U: 未定義，！: 特殊な条件を参照，*: 一般的場合
（X=C，N=結果の最上位ビット R_m，Z=演算結果が 0 のとき $\overline{R_m}\cdot\cdots\cdot\overline{R_0}$）

†1 記号の意味は表 6.1 参照

†2 商がオーバフローの場合は未定義．除数が 0 の場合は N=0，Z=1 に設定

†3 $0<$ データ レジスタ $<(\langle ea \rangle)$ のとき未定義

グには 0 が設定され，プロセッサは例外処理を開始する．

2 進化 10 進数演算命令では N フラグは意味をもたないので未定義である．

CHK 命令ではデータ レジスタ Dn の内容により，

（1） $Dn<0$ のとき $N=1$

（2） $Dn>(EA)$ のとき $N=0$

と N フラグが設定されトラップが生じプロセッサは例外処理を開始する．しかしデータ レジスタ Dn が (1)，(2) 以外のときには N フラグは未定義である．

MOVE to CCR/SR 命令および STOP 命令ではオペランドの対応するビットが各フラグに設定される．RTE, RTR 命令ではそれぞれシステム スタックの対応するビットが各フラグに設定される．

6.2.2 Z フラグ

Z (Zero) フラグは命令の実行結果がゼロかどうか示すフラグであり，結果がゼロのとき 1，そうでないとき 0 が設定される．一般的には，実行結果がゼロということは結果の各ビット R_i ($i=0\sim m$) が "0" ということであるから次式が成り立つ．

$$Z=\overline{R_m}\cdot\cdots\cdot\overline{R_0} \qquad (6\text{-}2)$$

拡張演算命令（ADDX, NEGX, SUBX 命令）および 2 進化 10 進数演算命令（ABCD, NBCD,

SBCD 命令）では，実行が完了した桁の演算に加え下位桁の状態も反映させなければならない．すなわち，拡張語全体としてあるいは2進化10進数全桁としてゼロかどうか判断する必要がある．したがって，

$$Z=Z\cdot\overline{R_m}\cdot\cdots\cdot\overline{R_0} \qquad (6\text{-}3)$$

となる．ここで右辺の Z は演算前すなわち下位桁の演算結果の Z フラグである．2進化10進数演算命令の最下位桁の実行に先立って，Z フラグと X フラグはクリアしておかなければならない．

大部分の命令では式（6-2）または（6-3）が成り立つが，DIVS, DIVU 命令，ビット操作命令，CHK 命令ならびに CCR を操作する命令は，これらにあてはまらない．

DIVS, DIVU 命令においては，商がゼロのとき Z フラグは1に設定される．商のオーバフローが生じた場合には Z フラグの値は未定義である．除数が0のときは Z フラグに1が設定されプロセッサは例外処理を開始する．

ビット操作命令（BCHG, BCLR, BSET, BTST 命令）では比較結果が Z フラグに反映された後に，ビット反転，ビットクリア，ビットセットが行われる．このため命令実行後のビットの値と Z フラグの値は必ずしも一致しない．

CHK 命令では Z フラグは未定義である．

6.2.3 V フラグ

V（Overflow）フラグは命令の算術演算においてオーバフローが生じたかどうか示すフラグである．V フラグには，オーバフローが生じたとき1が設定され，生じなかったとき0が設定される．ここで，オーバフローとは，演算結果が命令で指定したオペランドサイズ内で表現できない場合，すなわち符号ビットが破壊されてしまった場合をいう．

加算命令（ADD, ADDI, ADDQ, ADDX 命令）においてオーバフローが生じるのは図6.2で示すように2とおりの場合がある．

（1） 負の整数（デスティネーションオペランドの最上位ビット $S_m=1$）と負の整数（ソースオペランドの最上位ビット $D_m=1$）を加算した結果，符号ビット（実行結果の最上位ビット R_m）がクリアされた場合

$$V=S_m\cdot D_m\cdot\overline{R_m} \qquad (6\text{-}4)$$

（2） 正の整数と正の整数を加算した結果，符号ビットに1が設定された場合

$$V=\overline{S_m}\cdot\overline{D_m}\cdot R_m \qquad (6\text{-}5)$$

上記（1），（2）より加算命令におけるオーバフローは式（6-6）で表すことができる．

(a) $S_m \cdot D_m \cdot \overline{R_m}$　(b) $\overline{S_m} \cdot \overline{D_m} \cdot R_m$

図 6.2 加算命令においてオーバフローが生じる場合

(a) $S_m \cdot \overline{D_m} \cdot R_m$　(b) $\overline{S_m} \cdot D_m \cdot \overline{R_m}$

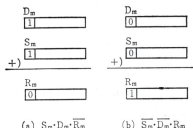
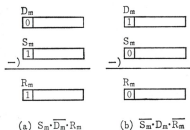



図 6.3 減算，比較命令においてオーバフローが生じる場合

$$V = S_m \cdot D_m \cdot \overline{R_m} + \overline{S_m} \cdot \overline{D_m} \cdot R_m \qquad (6\text{-}6)$$

減算命令（SUB, SUBI, SUBQ, SUBX 命令）および比較命令（CMP, CMPA, CMPI, CMPM 命令）におけるオーバフローも，図6.3に示すように，加算命令と同様に考えることができる．

減算，比較命令ではデスティネーションオペランドからソースオペランドが減算されることに注意を要する．図6.3から減算命令および比較命令を実行したとき V フラグは，(1) 正の整数−負の整数→符号ビット=1の場合，(2) 負の整数−正の整数→符号ビット=0の場合にセットされる．すなわち次式が成り立つことになる．

$$V = S_m \cdot \overline{D_m} \cdot R_m + \overline{S_m} \cdot D_m \cdot \overline{R_m} \qquad (6\text{-}7)$$

NEG, NEGX 命令は0からデスティネーションオペランドを減算する命令であり，減数がデスティネーションオペランドである点が減算，比較命令と異なる点である．この点に注意して V フラグの設定条件を求めると次式のとおりとなる（詳細省略）．

$$V = D_m \cdot R_m \qquad (6\text{-}8)$$

DIVS, DIVU 命令では商がオーバフローしたとき V フラグが設定される．

桁移動命令では ASL ($r \neq 0$) 以外の命令は V フラグを常に0とする．ASL 命令 ($r \neq 0$) では符号ビットに変化があった場合に V フラグに1が設定される．これには次の2とおりの場合がある．ここで D_m は桁移動前の最上位ビット，r は桁移動の数である．

（1） $D_m=1$ かつ $D_{m-1} \sim D_{m-r}$ のいずれかが0の場合

$$V = D_m \cdot (\overline{D_{m-1}} + \cdots + \overline{D_{m-r}}) \qquad (6\text{-}8)$$

（2） $D_m=0$ かつ $D_{m-1} \sim D_{m-r}$ のいずれかが1の場合

$$V = \overline{D_m} \cdot (D_{m-1} + \cdots + D_{m-r}) \qquad (6\text{-}9)$$

上記 (1), (2) より，ASL 命令 ($r \neq 0$) では次式に従って V フラグが設定される．

$$V = D_m \cdot (\overline{D_{m-1}} + \cdots + \overline{D_{m-r}}) + \overline{D_m} \cdot (D_{m-1} + \cdots + D_{m-r}) \tag{6-10}$$

この条件は桁移動によってデータの符号が変化した場合をチェックするためのものである.

このほか，命令によって0が設定されたり，未定義であったり，以前の値を保持していたりするが，これらについては表6.2を参照されたい.

6.2.4 C フラグ

C (Carry) フラグは命令の実行においてキャリィ/ボローが生じたかどうか示すフラグであり，キャリィ/ボローが生じた場合1が設定され，生じなかった場合0が設定される.

加算命令においてキャリィが生じるのは図6.4に示す3とおりの場合である．したがって，次式で示される条件によってCフラグが設定される.

$$C = S_m \cdot D_m + D_m \cdot \overline{R_m} + S_m \cdot \overline{R_m} \tag{6-11}$$

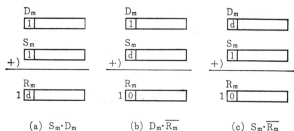



図 6.4　加算命令においてキャリィが生じる場合

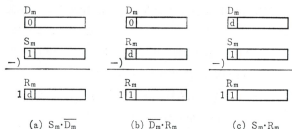



図 6.5　減算，比較命令においてボローが生じる場合

減算，比較命令においてボローが生じるのは図6.5に示す3とおりの場合である．したがって，次式で示される条件によってCフラグが設定される．

$$C=S_m\cdot\overline{D_m}+\overline{D_m}\cdot R_m+S_m\cdot R_m \quad (6\text{-}12)$$

NEG, NEGX 命令は0からデスティネーションオペランドを減算する命令であり，減数がデスティネーションオペランドである点を考慮し，上述した減算，比較命令と同様に考えると次式を得る．

$$C=D_m+R_m \quad (6\text{-}13)$$

2進化10進数演算命令では10進数のキャリィ/ボローがCフラグに反映される．

シフト回数が0の桁移動命令において，ROXL, ROXR 命令ではCフラグにXフラグの内容がセットされその他のシフト/ローテート命令ではC=0となる．1回以上シフトまたはローテートする桁移動命令では最後にシフトアウトされたビットがCフラグに反映される．すなわち，右シフトをr回行った後には

$$C=D_{r-1}$$

となり，左シフトをr回行った後には

$$C=D_{m-r+1}$$

が設定される．

このほか，Cフラグが0に設定されたり，未定義であったり，以前の値を保持したりする命令については表6.2を参照されたい．

6.2.5 X フラグ

Xフラグは拡張演算および拡張形桁移動を行うときに使用する．Xフラグが変化するときは常にX=Cとなる．しかしCフラグが変化したとき常にXフラグが影響を受けるわけではない．

Xフラグの設定条件は表6.2を参照されたい．

7 命令実行時間

この章では68000の命令実行時間について解説する.

7.1 命令実行時間の内訳

一般に命令実行時間は図7.1 (a) に示すごとく (1) 命令フェッチ時間: t_F, (2) 実効アドレス計算時間: t_{EA}, (3) 命令処理時間: t_{EX} の総和で表される. ところが68000では, 2章にも述べたように, 命令プリフェッチ機能を有するために, 基本的には命令実行手順

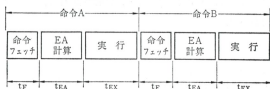



図 7.1 命令実行時間の内訳

は図7.1 (b) に示すようになる．すなわち命令フェッチ（時間 t_F）が，オペランドの実効アドレス計算（時間 t_{EA}）または命令処理（時間 t_{EX}）と平行して行われるため実効的な命令実行時間は $t_{EA}+t_{EX}$ で表される．

68000 には，4章で述べたように，多様なアドレス形式があり，各々のアドレス形式で実効アドレス計算時間 t_{EA} が異なる．また，各命令はその処理時間 t_{EX} が異なるため，次節以降で述べる具体的な命令実行時間では t_{EA} と t_{EX} を別々に解説する．また，メモリアクセスは十分高速であるものとし，ウエート状態（Sw）はないものと仮定した．

7.2 実効アドレス計算時間

ここでは時間を ns の代りにクロックサイクル数で示す．クロック サイクル数で表現した理由は，68000 は動作クロック周波数が 4 MHz 版から 6 MHz～12 MHz の品種が用意されているために ns で表現するより一般性を有しているためである．ここでサイクル数は動作クロックの周期を1単位として表している．したがって，たとえば4クロック サイクルは 8 MHz 版 68000 では 4×125 ns＝500 ns を意味する．

68000 は4章で述べたように12種の実効アドレス形式をもっている．それら各実効アドレス計算に要するクロック サイクル数は表7.1に示される．なお，このサイクル数で処理される内容には次の2つが含まれる．

（1） 実効アドレス計算

表 7.1 実効アドレス計算処理時間（サイクル数）

アドレス形式		オペランドの長さ	
		バイト/ワード	ロング ワード
Dn	データ レジスタ直接形式	0 (0/0)	0 (0/0)
An	アドレス レジスタ直接形式	0 (0/0)	0 (0/0)
An@	アドレス レジスタ間接形式	4 (1/0)	8 (2/0)
An@+	ポストインクリメント アドレス レジスタ間接形式	4 (1/0)	8 (2/0)
An@−	プリデクリメント アドレス レジスタ間接形式	6 (1/0)	10 (2/0)
An@d	ディスプレースメント付アドレス レジスタ間接形式	8 (2/0)	12 (3/0)
An@(d, ix)	インデックス付アドレス レジスタ間接形式	10 (2/0)	14 (3/0)
×××.W	短絶対アドレス形式	8 (2/0)	12 (3/0)
×××.L	長絶対アドレス形式	12 (3/0)	16 (4/0)
PC@(d)	ディスプレースメント付プログラムカウンタ相対形式	8 (2/0)	12 (3/0)
PC@(d, ix)	インデックス付プログラムカウンタ相対形式	10 (2/0)	14 (3/0)
# ×××	イミディエイト形式	4 (1/0)	8 (2/0)

注） (a/b)，a；リード サイクル数，b；ライト サイクル数

（2） メモリからのオペランド フェッチ

同表ではオペランドがバイト（8ビット）またはワード（16ビット）の場合とロングワード（32ビット）の場合とに分けて示されている．両者とも実効アドレス計算処理に要するサイクル数は同じであるが，表中で差があるのはオペランド フェッチ処理の差に起因する．すなわち，1ワードのフェッチ（メモリ リード）には4サイクルを要しロング ワードのフェッチには8サイクル要するためである．また，同表において（ ）中には（リード サイクル数/ライト サイクル数）が示される．これらはいずれも全サイクル数の中に含まれるサイクル数である．実効アドレス計算処理においてライト サイクル数がいずれの場合も0であるのはその処理の性質から当然のことである．

7.3 オペレーション実行処理時間

以下に68000の各命令の実行処理時間 t_{EX} をサイクル数で示す．なお，データ MOVE 命令に関してだけは，68000においては種々の実効アドレス形式を有する2つのオペランドを指定し得ることから単純に表7.1に示した t_{EA} と t_{EX} を加えた値では表現しにくい．このため MOVE 命令についてはソース オペランドとデスティネーションオペランドの実効アドレス形式の組合せによって処理時間がわかるように $t_{EA}+t_{EX}$ の全時間で表現した．

7.3.1 データ転送命令時間（実効アドレス計算処理時間を含む）

表7.2はオペランド サイズがバイト長またはワード長の場合のデータ転送命令の実行時間を示している．また，表7.3はオペランド サイズがロング ワードの場合のデータ転送命令の実行時間を示している．これらの表をよくみると表の対角線を軸としてほぼ対称形の表になっている．しかし An@−（プリデクリメント アドレス レジスタ間接）モードが指定された場合に限ってこの対称形が保たれない．たとえばソース アドレス＝An@−，デスティネーション オペランド＝Dn の場合は命令実行時間が10サイクルであるのに対して，逆に，ソース オペランド＝Dn，デスティネーション アドレス＝An@− の場合は8サイクルですむ．この理由はソース アドレスに An@− モードが指定された場合は命令実行の最初にアドレス レジスタをデクリメントする2サイクルの操作が入ることによる．デスティネーション オペランドに An@− モードが指定された場合のアドレス レジスタのデクリメント操作は命令フェッチ中に実行し得るためデクリメント操作のため

表 7.2 オペレーション サイズがバイト，ワードのときの MOVE 命令実行時間（サイクル数）

Sアドレス ＼ Dアドレス	Dn	An	An@	An@+	An@−	An@(d)	An@(d, ix)	××× .W	××× .L
Dn	4 (1/0)	4 (1/0)	8 (1/1)	8 (1/1)	8 (1/1)	12 (2/1)	14 (2/1)	12 (2/1)	16 (3/1)
An	4 (1/0)	4 (1/0)	8 (1/1)	8 (1/1)	8 (1/1)	12 (2/1)	14 (2/1)	12 (2/1)	16 (3/1)
An@	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	16 (3/1)	18 (3/1)	16 (3/1)	20 (4/1)
An@+	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	16 (3/1)	18 (3/1)	16 (3/1)	20 (4/1)
An@−	10 (2/0)	10 (2/0)	14 (2/1)	14 (2/1)	14 (2/1)	18 (3/1)	20 (3/1)	18 (3/1)	22 (4/1)
An@(d)	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	20 (4/1)	22 (4/1)	20 (4/1)	24 (5/1)
An@(d, ix)	14 (3/0)	14 (3/0)	18 (3/1)	18 (3/1)	18 (3/1)	22 (4/1)	24 (4/1)	22 (4/1)	26 (5/1)
××× .W	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	20 (4/1)	22 (4/1)	20 (4/1)	24 (5/1)
××× .L	16 (4/0)	16 (4/0)	20 (4/1)	20 (4/1)	20 (4/1)	24 (5/1)	26 (5/1)	24 (5/1)	28 (6/1)
PC@(d)	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	20 (4/1)	22 (4/1)	20 (4/1)	24 (5/1)
PC@(d, ix)	14 (3/0)	14 (3/0)	18 (3/1)	18 (3/1)	18 (3/1)	22 (4/1)	24 (4/1)	22 (4/1)	26 (5/1)
# ×××	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	16 (3/1)	18 (3/1)	16 (3/1)	20 (4/1)

表 7.3 オペレーション サイズがロング ワードの時の MOVE 命令実行時間（サイクル数）

Sアドレス ＼ Dアドレス	Dn	An	An@	An@+	An@−	An@(d)	An@(d, ix)	××× .W	××× .L
Dn	4 (1/0)	4 (1/0)	12 (1/2)	12 (1/2)	12 (1/2)	16 (2/2)	18 (2/2)	16 (2/2)	20 (3/2)
An	4 (1/0)	4 (1/0)	12 (1/2)	12 (1/2)	12 (1/2)	16 (2/2)	18 (2/2)	16 (2/2)	20 (3/2)
An@	12 (3/0)	12 (3/0)	20 (3/2)	20 (3/2)	20 (3/2)	24 (4/2)	26 (4/2)	24 (4/2)	28 (5/2)
An@+	12 (3/0)	12 (3/0)	20 (3/2)	20 (3/2)	20 (3/2)	24 (4/2)	26 (4/2)	24 (4/2)	28 (5/2)
An@−	14 (3/0)	14 (3/0)	22 (3/2)	22 (3/2)	22 (3/2)	26 (4/2)	28 (4/2)	26 (4/2)	30 (5/2)
An@(d)	16 (4/0)	16 (4/0)	24 (4/2)	24 (4/2)	24 (4/2)	28 (5/2)	30 (5/2)	28 (5/2)	32 (6/2)
An@(d, ix)	18 (4/0)	18 (4/0)	26 (4/2)	26 (4/2)	26 (4/2)	30 (5/2)	32 (5/2)	30 (5/2)	34 (6/2)
××× .W	16 (4/0)	16 (4/0)	24 (4/2)	24 (4/2)	24 (4/2)	28 (5/2)	30 (5/2)	28 (5/2)	32 (6/2)
××× .L	20 (5/0)	20 (5/0)	28 (5/2)	28 (5/2)	28 (5/2)	32 (6/2)	34 (6/2)	32 (6/2)	36 (7/2)
PC@(d)	16 (4/0)	16 (4/0)	24 (4/2)	24 (4/2)	24 (4/2)	28 (5/2)	30 (5/2)	28 (5/2)	32 (6/2)
PC@(d, ix)	18 (4/0)	18 (4/0)	26 (4/2)	26 (4/2)	26 (4/2)	30 (5/2)	32 (5/2)	30 (5/2)	34 (6/2)
# ×××	12 (3/0)	12 (3/0)	20 (3/2)	20 (3/2)	20 (3/2)	24 (4/2)	26 (4/2)	24 (4/2)	28 (5/2)

表 7.4　標準命令実行処理時間（サイクル数）

命　令	オペランドアドレス ＼ オペレーションサイズ	Sアドレス=(EA) Dアドレス=An	Sアドレス=(EA) Dアドレス=Dn	Sアドレス=Dn Dアドレス=(EA)
ADD	バイト，ワード ロング ワード	8 (1/0) 6 (1/0)	4 (1/0) 6 (1/0)	8 (1/1) 12 (1/2)
AND	バイト，ワード ロング ワード	— —	4 (1/0) 6 (1/0)	8 (1/1) 12 (1/2)
CMP	バイト，ワード ロング ワード	6 (1/0) 6 (1/0)	4 (1/0) 6 (1/0)	— —
DIVS	—	—	158 (1/0)MAX	—
DIVU	—	—	140 (1/0)MAX	—
EOR	バイト，ワード ロング ワード	— —	4 (1/0) 8 (1/0)	8 (1/1) 12 (1/2)
MULS	—	—	70 (1/0)MAX	—
MULU	—	—	70 (1/0)MAX	—
OR	バイト，ワード ロング ワード	— —	4 (1/0) 6 (1/0)	8 (1/1) 12 (1/2)
SUB	バイト，ワード ロング ワード	8 (1/0) 6 (1/0)	4 (1/0) 6 (1/0)	8 (1/1) 12 (1/2)

注）命令実行時間は上表の値に実効アドレス生成時間を加える必要あり
D=デスティネーション，S=ソース

の2サイクルは実効的な時間としては現れない．

7.3.2　標準命令実行時間

表7.4に ADD, AND 命令等の標準命令の命令処理時間を示す．これらの命令が使用するアドレス形式は各種あるが，共通していることは（レジスタ，(EA)）間の処理であるということである．すなわち，データ転送命令のようにソース オペランド，デスティネーション オペランドが共にメモリ上にあることはない．必ず一方はレジスタが指定される．

なお，表7.4に示したサイクル数は命令の処理サイクル，結果の格納サイクル，および次の命令のフェッチ サイクルを含んでいる．したがって命令の全実行時間を求めるためには (EA) を計算するためのアドレス生成サイクル数 t_{EA} を表7.1より求めて表7.4の値に加えなければならない．

たとえば次の命令の実行時間を求めてみる．

ADD .L D1, (A1)

デスティネーション オペランドのアドレス形式はアドレス レジスタ間接形式であり, オペランド サイズがロング ワードである. したがって, 実効アドレス計算時間 t_{EA} は表7.1より8クロック サイクルとなる. また, ADD 命令処理時間 t_{EX} は, ソース オペランドが D1, デスティネーション オペランドが (EA), オペランド サイズがロング ワードであるから, 表7.4より t_{EX}=12 クロック サイクルとなる. 以上より上記 ADD 命令の実行時間は 8+12=20 クロック サイクルとなる.

7.3.3 イミディエイト命令実行時間

表7.5にイミディエイト命令の実行処理時間を示す. この表に示される処理時間に含まれる内容はイミディエイト オペランドのフェッチ サイクル, 命令の実行処理サイクル, 結果の格納サイクルおよび次の命令のプリフェッチ サイクルである. したがってイミディ

表 7.5 イミディエイト命令実行処理時間 (サイクル数)

命 令	オペランド サイズ ＼ オペランド	Sアドレス=イミディエイト Dアドレス=Dn	Sアドレス=イミディエイト Dアドレス=An	Sアドレス=イミディエイト Dアドレス=(EA)
ADDI	バイト, ワード	8 (2/0)	—	12 (2/1)
	ロング ワード	16 (3/0)	—	20 (3/2)
ADDQ	バイト, ワード	4 (1/0)	8 (1/0)	8 (1/1)
	ロング ワード	8 (1/0)	8 (1/0)	12 (1/2)
ANDI	バイト, ワード	8 (2/0)	—	12 (2/1)
	ロング ワード	16 (3/0)	—	20 (3/1)
CMPI	バイト, ワード	8 (2/0)	8 (2/0)	8 (2/0)
	ロング ワード	14 (3/0)	14 (3/0)	12 (3/0)
EORI	バイト, ワード	8 (2/0)	—	12 (2/1)
	ロング ワード	16 (3/0)	—	20 (3/2)
MOVEQ	ロング ワード	4 (1/0)	—	—
ORI	バイト, ワード	8 (2/0)	—	12 (2/1)
	ロング ワード	16 (3/0)	—	20 (3/2)
SUBI	バイト, ワード	8 (2/0)	—	12 (2/1)
	ロング ワード	16 (3/0)	—	20 (3/2)
SUBQ	バイト, ワード	4 (1/0)	8 (1/0)	8 (1/1)
	ロング ワード	8 (1/0)	8 (1/0)	12 (1/2)

注) 命令実行時間は上表の値に実効アドレス生成時間を加える必要あり
S=ソース, D=デスティネーション

エイト命令の全実行時間を求めるためには（EA）を計算するための実効アドレス計算時間を表7.1より求め，表7.5の値に加えなければならない．

デスティネーション オペランドがDnまたはAnである場合は，デスティネーション オペランドの実効アドレス計算に要する時間は表7.1から明らかなように0である．したがって表7.5に示した値がすなわち命令実行の全時間に等しい．デスティネーション オペランド=(EA）である場合は，種々の実効アドレス モードをとり得るので表7.1より実効アドレス生成時間を表7.5の値に加えなければならない．

さて，一例としてADDI命令について述べてみよう．ADDI命令はイミディエイト値とデスティネーション オペランドの内容を加えて，その結果をデスティネーション アドレスに格納する命令である．オペランドがバイト長またはワード長でデスティネーション オペランドがDnの場合を考える．実行処理時間を表7.5から求めると8(2/0）となっている．すなわち命令処理に8サイクルを要する．デスティネーション オペランドはDnであるため，そのアドレス生成時間は表7.1より0サイクルである．すなわち，この命令の全実行時間は8+0=8サイクルである．

7.3.4 シングル オペランド命令実行時間

表7.6にシングル オペランド命令の実行処理時間を示す．この表に示される処理時間に含まれる処理内容はオペランドのフェッチ，命令の処理，および次の命令プリフェッチである．したがって命令実行の全時間を求めるためには表7.1に示されたオペランド アドレスの計算時間を表7.6に示された値に加えなければならない．シングル オペランド命令はCLR, NBCD, NEG命令など8種の命令がある．その特徴はすべて2バイト（1ワード）命令であることと，オペランドには他の命令と異なりデスティネーション オペランドしかもたないことである．オペランドがレジスタ（DnまたはAn）の場合には，表7.6の値に加えるべきアドレス計算時間は表7.1より0サイクルであるため，表7.6の値がそのまま命令の全実行時間となる．一方，オペランドがメモリの場合には，命令の全実行時間は表7.6の値に表7.1に示したアドレス計算時間を各々実効アドレス形式に従って求めて加えなければならない．

7.3.5 シフト/ローテート命令実行時間

表7.7にシフト/ローテート命令の実行処理時間を示す．この表に示す処理時間に含まれる内容はオペランド フェッチ，シフト/ローテートの処理，結果の格納および，次の命

表 7.6 シングルオペランド命令実行処理時間（サイクル数）

命　令	オペランド／オペランドサイズ	オペランド＝レジスタ（Dn または An）	オペランド＝メモリ（EA）
CLR	バイト，ワード	4 (1/0)	8 (1/1)
	ロング ワード	6 (1/0)	12 (1/2)
NBCD	バイト	6 (1/0)	8 (1/1)
NEG	バイト，ワード	4 (1/0)	8 (1/1)
	ロング ワード	6 (1/0)	12 (1/2)
NEGX	バイト，ワード	4 (1/0)	8 (1/1)
	ロング ワード	6 (1/0)	12 (1/2)
NOT	バイト，ワード	4 (1/0)	8 (1/1)
	ロング ワード	6 (1/0)	12 (1/2)
Scc	バイト，ワード	4 (1/0)	8 (1/1)
	ロング ワード	6 (1/0)	8 (1/1)
TAS	バイト	4 (1/0)	10 (1/1)
TST	バイト，ワード	4 (1/0)	4 (1/0)
	ロング ワード	4 (1/0)	4 (1/0)

注） 命令実行時間は上表の値に実効アドレス生成時間を加える必要あり

表 7.7 シフト/ローテート命令実行処理時間（サイクル数）

命　令	オペランド／オペランドサイズ	オペランド＝レジスタ	オペランド＝メモリ（EA）
ASR, ASL	バイト，ワード	6+2n* (1/0)	8 (1/1)
	ロング ワード	8+2n (1/0)	—
LSR, LSL	バイト，ワード	6+2n (1/0)	8 (1/1)
	ロング ワード	8+2n (1/0)	—
ROR, ROL	バイト，ワード	6+2n (1/0)	8 (1/1)
	ロング ワード	8+2n (1/0)	—
ROXR, ROXL	バイト，ワード	6+2n (1/0)	8 (1/1)
	ロング ワード	8+2n (1/0)	—

注） 命令実行時間は上表の値に実効アドレス生成時間を加える必要あり
* n は桁移動の数を表す

令のプリフェッチである．したがってオペランド アドレスの計算時間を表 7.1 により求め，表 7.7 に示した値に加えて命令の全実行時間を求めなければならない．シフト/ローテート命令の機能はオペランドがレジスタの場合には 1 命令で n ビットのシフトまたはローテートが可能であるが，一方オペランドがメモリの場合には 1 命令で実行し得るシフ

ト/ローテートビット数は1ビットだけである．1ビットのシフトには2クロックサイクルを必要とすることから，表7.7においてオペランドがレジスタの場合には +2n サイクルが加えられている．

たとえば次の命令の実行時間は，$t_{EA}=0$，$t_{EX}=6+2\times8=22$ であるから22クロックサイクルとなる．

```
ASL .W  #8, D1
```

7.3.6 ビット操作命令実行時間

表7.8にビット操作命令の実行処理時間を示す．この表に示す値はオペランドフェッチ，命令の処理，結果の格納，および次の命令のプリフェッチを含んでいる．したがって命令の全実行時間を求めるためにはオペランドアドレスの計算時間を表7.1から求めて表7.8の値に加えなければならない．表7.8中のダイナミック形とは操作すべきオペランドのビット位置が命令で指定されるレジスタに貯えられている場合を示し，一方スタティック形とは操作すべきオペランドのビット位置が直接命令で指定される場合を示している．

7.3.7 ブランチ関係命令実行時間

表7.9にブランチ関係の命令実行処理時間を示す．この表には，ブランチ先のアドレス（ディスプレースメント付プログラムカウンタ相対形式）の計算に要する時間が含まれている．したがって Bcc, BRA, BSR, DBcc 命令に関しては本表に示された値が命令の全実行時間を示している．また TRAP, TRAPV 命令についてもベクタ生成時間は本表の値に含まれているので命令の全実行時間を示している．しかし CHK 命令ではブランチ先のアドレスとして各種の実効アドレス形式を有しているので命令の全実行時間を求めるためには表7.9に示される値に，表7.1より求めた実効アドレス計算時間を加えなければならない．さて，表7.9はブランチが成立した場合と不成立の場合について分けて示している．BRA, BSR, TRAP 命令は無条件ブランチであるので不成立の欄は無意味である．68000では命令のプリフェッチ機能を有しているため，ブランチが成立したときには先行してプリフェッチしていた命令が無駄になる．したがってブランチ条件が成立した場合には命令処理時間がブランチ条件不成立の場合に比べて長くなると考えられる．この状況が表7.9の Bcc 命令，ディスプレースメント＝バイトの場合の実行時間に現れている．しかし，この弊害を最小化するために68000ではブランチアドレスをマイクロプログラ

表 7.8 ビット操作命令実行処理時間（サイクル数）

命令	ビット位置 / オペランド / オペランドサイズ	ダイナミック レジスタ	ダイナミック メモリ(EA)	スタティック レジスタ	スタティック メモリ(EA)
BCHG	バイト	—	8 (1/1)	—	12 (2/1)
	ロング ワード	8(1/0)MAX	—	12(2/0)MAX	—
BCLR	バイト	—	8 (1/1)	—	12 (2/1)
	ロング ワード	10(1/0)MAX	—	14(2/0)MAX	—
BSET	バイト	—	8 (1/1)	—	12 (2/1)
	ロング ワード	8(1/0)MAX	—	12(2/0)MAX	—
BTST	バイト	—	4 (1/0)	—	8 (2/0)
	ロング ワード	6(1/0)MAX	—	10(2/0)MAX	—

注） 命令実行時間は上表の値に実効アドレス成生時間を加える必要あり

表 7.9 ブランチ関係命令実行処理時間（サイクル数）

命令	条件成立の有無 / ディスプレースメント	トラップまたはブランチ条件成立	トラップまたはブランチ条件不成立
Bcc	バイト	10 (2/0)	8 (1/0)
	ワード	10 (2/0)	12 (2/0)
BRA	バイト	10 (2/0)	—
	ワード	10 (2/0)	—
BSR	バイト	18 (2/2)	—
	ワード	18 (2/2)	—
DBcc	CC true	—	12 (2/0)
	CC false	10 (2/0)	14 (3/0)
CHK	—	40 (5/3)+$t_{EA\ MAX}$	10 (1/0)+t_{EA}
TRAP	—	34 (4/3)	—
TRAPV	—	34 (5/3)	4 (1/0)

注） 命令実行時間は上表の値に実効アドレス生成時間を加える必要あり

ムで先行して計算するように配慮されているので，場合によってはブランチ条件が成立した場合の方が不成立の場合より高速に処理できるのである．その例が Bcc 命令においてディスプレースメント＝ワードの場合である．Bcc 命令でディスプレースメントがワード長の場合にはロング ワード命令（4バイト命令）となるためプリフェッチの効果が生きないことがその理由である．DBcc 命令の実行時間がブランチ条件成立の場合が不成立の場合よりも高速になっているのも同じ理由による．

表 7.10 ジャンプ関係命令実行時間（サイクル数）

命令 ＼ E.A.	An@	An@+	An@−	An@ (d)	An@ (d, ix)	×××.W	×××.L	PC@ (d)	PC@ (d, ix)
JMP	8(2/0)	—	—	10(2/0)	14(3/0)	10(2/0)	12(3/0)	10(2/0)	14(3/0)
JSR	16(2/2)	—	—	18(2/2)	22(2/2)	18(2/2)	20(3/2)	18(2/2)	22(2/2)

E.A.＝実効アドレス形式

表 7.11 実効アドレス生成関係命令実行時間（サイクル数）

命令 ＼ E.A.	An@	An@+	An@−	An@ (d)	An@ (d, ix)	×××.W	×××.L	PC@ (d)	PC@ (d, ix)
LEA	4(1/0)	—	—	8(2/0)	12(2/0)	8(2/0)	12(3/0)	8(2/0)	12(2/0)
PEA	12(1/2)	—	—	16(2/2)	20(2/2)	16(2/2)	20(3/2)	16(2/2)	20(2/2)

E.A.＝実効アドレス形式

7.3.8 ジャンプ関係命令実行時間

表7.10にジャンプ関係命令（JMP, JSR 命令）の実行時間を示す．ここに示された値はジャンプ先アドレスの計算，ジャンプの実行およびジャンプ先番地に格納されている次の命令プリフェッチの時間を含んでいる．すなわち，実効アドレス計算時間が表7.10に含まれているため表7.1に示した実効アドレス計算時間を加えなくても，表7.10の値はジャンプ命令の全実行時間を表している．

7.3.9 実効アドレス生成関係命令実行時間

表7.11に実効アドレス生成関係命令（LEA, PEA 命令）の実行時間を示す．ここに示された値には実効アドレス計算時間が含まれるので改めて表7.1の値を加える必要はない．すなわち，表7.11の値は命令の全実行時間を表している．

7.3.10 マルチレジスタ関係命令実行時間

表7.12にマルチレジスタ関係命令の実行時間を示す．マルチレジスタ関係の命令は MOVEM 命令のみである．同表において n は移動すべきオペランドの数を示している．ワード長オペランドのときにはメモリ-レジスタ間移動に要するサイクル数は1オペランドにつき4サイクル，一方ロングワードオペランドのときには1オペランド当り8サイクルを要する．表7.12に示される値には実効アドレス生成のための時間が含まれているので，改めて表7.1から実効アドレス生成時間を求め表7.12の値に加える必要はない．すなわち，表7.12の値は MOVEM 命令の全実行時間を表している．M→R の場合と R→M

表 7.12　マルチレジスタ関係命令実行時間（サイクル数）

命令	E.A. / O.S.	An@	An@+	An@−	An@(d)	An@ (d, ix)	×××.W	×××.L	PC@(d)	PC@ (d, ix)
MOVEM	ワード	12+4n	12+4n	—	16+4n	18+4n	16+4n	20+4n	16+4n	18+4n
(M→R)		(3+n/0)	(3+n/0)		(4+n/0)	(4+n/0)	(4+n/0)	(5+n/0)	(4+n/0)	(4+n/0)
	ロング ワード	12+8n	12+8n	—	16+8n	18+8n	16+8n	20+8n	16+8n	18+8n
		(2+2n/0)	(3+n/0)		(4+2n/0)	(4+2n/0)	(4+2n/0)	(5+2n/0)	(4+2n/1)	(4+2n/0)
MOVEM	ワード	8+4n	—	8+4n	12+4n	14+4n	12+4n	16+4n	—	—
(R→M)		(4/n)		(2/n)	(3/n)	(3/n)	(3/n)	(4/n)		
	ロング ワード	8+8n	—	8+8n	12+8n	14+8n	12+8n	16+8n	—	—
		(2/2n)		(2/2n)	(3/2n)	(3/2n)	(3/2n)	(4/2n)		

O.S.＝オペランドサイズ，E.A.＝実効アドレス形式

表 7.13　倍精度演算命令実行時間（サイクル数）

命令	O.O. / O.S.	S オペランド＝Dn D オペランド＝Dn	S オペランド＝メモリ D オペランド＝メモリ
ADDX	バイト，ワード	4 (1/0)	18 (3/1)
	ロング ワード	8 (1/0)	30 (5/2)
SUBX	バイト，ワード	4 (1/0)	18 (3/1)
	ロング ワード	8 (1/0)	30 (5/2)
ABCD	バイト	6 (1/0)	18 (3/1)
SBCD	バイト	6 (1/0)	18 (3/1)
CMPM	バイト，ワード	—	12 (3/0)
	ロング ワード	—	20 (5/0)

O.S.＝オペランドサイズ，O.O.＝演算オペランド形式，S＝ソース，D＝デスティネーション

の場合の実行時間に4サイクルの差がある．たとえばワードオペランド，実効アドレス＝An@ の場合を考えると M→R のときは（12+4n）サイクルであるが，R→M のときは（8+4n）サイクルである．4n サイクルは n ワード分のメモリリードまたはライトに要するサイクル数である．差の4サイクル分（12−8=4）は 68000 のアーキテクチャにかかわるもので，高速動作実現のために 68000 はメモリデータの先読みを実行しており，n ワード移動に対して実際は n+1 ワード分メモリから MPU 内にリードしていることによるものである．

7.3.11 倍精度演算命令実行時間

表7.13に倍精度演算命令の実行時間を示す．表7.13の示す値は2つのオペランド フェッチ，演算，結果の格納，および次の命令のとり込みの各時間のほかに2つのオペランドの実効アドレス計算時間をも含んでいる．オペランドの実効アドレスの生成時間を含めて表現した理由はこれらの命令では2つのメモリ オペランドが扱われるからである．表7.1に示したオペランドの実効アドレス生成時間は1つのオペランド アドレスを計算する時間を表している．2つのメモリ オペランドを有する命令の実行時間は表7.1に示されたメモリ オペランドが2つあるから単純に2倍して実行処理時間に加えるという計算で簡単に求めることはできない．

7.3.12 その他の命令実行時間

表7.14に前項までに述べた各種命令以外の残りの命令の実行時間をまとめて示す．表

表 7.14　その他の命令実行処理時間（サイクル数）

命　令	O.S. ＼ O.	レジスタ	メモリ	レジスタ→メモリ	メモリ→レジスタ
MOVE from SR	—	6 (1/0)	8 (1/1)	—	—
MOVE to CCR	—	12 (2/0)	12 (2/0)	—	—
MOVE to SR	—	12 (2/0)	12 (2/0)	—	—
MOVEP	ワード	—	—	16 (2/2)	16 (4/0)
	ロング ワード	—	—	24 (2/4)	24 (6/0)
EXG	—	6 (1/0)	—	—	—
EXT	ワード	4 (1/0)	—	—	—
	ロング ワード	4 (1/0)	—	—	—
LINK	—	16 (2/2)	—	—	—
MOVE from USP	—	4 (1/0)	—	—	—
MOVE to USP	—	4 (1/0)	—	—	—
NOP	—	4 (1/0)	—	—	—
RESET	—	132 (1/0)	—	—	—
RTE	—	20 (5/0)	—	—	—
RTR	—	20 (5/0)	—	—	—
RTS	—	16 (4/0)	—	—	—
STOP	—	4 (0/0)	—	—	—
SWAP	—	4 (1/0)	—	—	—
UNLK	—	12 (3/0)	—	—	—

注）命令実行時間は上表の値に実効アドレス生成時間を加える必要あり
O.S.＝オペランド サイズ，O.＝オペランド形式

7.14に示した値にはオペランドの実効アドレス生成時間は含まれていないので，命令実行時間を求めるためには表7.1に示した実効アドレス生成時間を加えなければならない．オペランドがレジスタのみの場合の実効アドレス生成時間は0であるから実質的には表7.1に示した値が実行時間になる．しかし，オペランドがメモリの場合は表7.1から実効アドレス生成時間を求めなければならない．MOVE from SR, MOVE to SR, MOVE to CCR の各命令はメモリ オペランドとして各種実効アドレス モードを指定し得る．MOVEP 命令については指定されるメモリ オペランド実効アドレス形式は An@(d) 形式のみである．

7.3.13 例外処理時間

表7.15に例外処理に必要なクロック数を示す．例外処理は命令の実行ではないがその処理時間を示しておく．例外処理の内容については8章で説明する．表7.15で示されるクロック サイクル数はプロセッサが例外処理を開始してから例外処理の新プログラムの読出しまでの時間である．

表 7.15 例外処理時間（サイクル数）

例外処理	時　間
リセット	34 (6/0)
アドレス エラー	50 (4/7)
バス エラー	50 (4/7)
割込み	44 (5/3)*
不当命令	34 (4/3)
特権命令	34 (4/3)
トレース	34 (4/3)

* 割込みアクノレッジ バス サイクルには，4 クロックサイクルを要すると仮定している

8　例 外 処 理

この章では68000のプロセッサ処理状態，プログラム実行状態，例外処理について説明する．これらは他のマイクロコンピュータにはみられない進んだアーキテクチャであり，68000を理解する上で重要である．

8.1　プロセッサ処理状態

68000が動作をしている場合，その動作は通常状態，例外状態，ホールト状態の3状態に分類される．これらの状態を総称してプロセッサ処理状態という．

通常状態とはプログラムを実行している状態である．この状態では，メモリ参照によって命令やオペランド等を読み出し，演算を行った後にその結果を内部レジスタあるいはメモリに格納する．これらの動作は連続的に実行されるが，その例外として STOP 命令（17章，特権命令参照）の実行によるストップ状態がある．STOP 命令を実行すると，この命令によってステータス レジスタにセットされた割込みレベルより高いレベルの外部割込みが発生するまで，プロセッサはストップする．このストップ状態は STOP 命令を正常に実行した結果である．そのため，後述するホールト状態のようなシステムの回路要求すなわち，外部からの制御信号の入力による停止とは区別され，通常状態として分類する．

例外状態とは，外部割込み，トラップ命令の実行，バスエラー，アドレスエラー等の例外処理要因によって起動される処理を実行している状態である．例外処理で実行される動作に関しては8.3節で詳細に説明する．

ホールト状態とはシステムの回路的要求によってプロセッサが停止する状態である．この回路的要求には，3章でも説明したようにホールト信号入力ピンのアサートによる場合と2重バス障害による場合とがある．前者は，68000に接続された外部デバイスが何らかの重大な故障を起した場合，ホールト信号を入力してプロセッサを停止させる場合であ

る．後者は，例外処理実行中にメモリ参照ができないためその処理が続行できないという重大な故障が発生した場合，プロセッサを停止させるものである．

プロセッサ処理状態の遷移を図8.1にまとめる．通常状態から例外状態への遷移は，前述のように例外処理を必要とする要因の発生によって行われる．その例外処理が終了すると再び通常状態に遷移する．通常状態，例外状態においてホールト信号入力ピンがアサートされるとホールト状態に遷移する．ホールト信号入力ピンをネゲートすると，ホールト状態となる前の状態へ遷移する．また，例外処理中にメモリの参照が不可能となり2重バス障害（通常状態では2重バス障害は発生しない）が発生した場合には，例外状態からホールト状態に遷移する．その後，プロセッサを再び起動させるためには，$\overline{\text{RES}}$ ピンをアサートして，例外状態へ遷移させる必要がある．

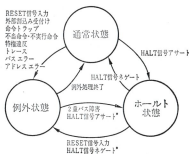



図 8.1 プロセッサの処理状態の遷移

8.2 プログラム実行状態

68000では，前節8.1で説明した通常状態において，実行するプログラムを一般のプログラムとシステムプログラムとに区別できるようにしている．そのためにプログラム実行状態† をユーザ状態とスーパバイザ状態という2種類の状態に区別する．この2種類の状態は，ステータス レジスタ内のSビット（ビット番号13）によって選択される．すなわちSビットが“0”の場合にはユーザ状態であり，“1”の場合にはスーパバイザ状態である．

システムの信頼性を考えた場合，一般のプログラムの実行と，システムを管理するプログラム（OSと称されるシステム プログラム）の実行とを区別することが重要である．一般のプログラムの実行において，システム全体に影響を与えるような動作が簡単に行われると，いくつかのプログラムを並列させて実行する場合には大きな障害となりかねない．そこで，一般のプログラムで実行できる機能は制限し，システムを管理する機能等は特殊な状態で実行するシステム プログラムに任せる方式がシステム全体の信頼性を向上させるためには有効である．

† 従来の68000解説書ではプログラム実行状態（Privilege State）を特権状態と訳してきた．しかし，この特権状態という表現はスーパバイザ状態（この状態においてのみ特権命令を使用できる）と混同される可能性があるため，本書では，プログラム実行状態という表現を用いることにした．

スーパバイザ状態は，プログラム実行状態の2つの状態のうちの上位の状態に相当する．一般にこの状態でシステム プログラムが実行される．システムを管理する上で有効な命令である特権命令（17章参照）は，スーパバイザ状態においてのみ使用可能である．

一方，ユーザ状態は，プログラム実行状態における下位の状態に相当する．この状態で前述の特権命令を使用しようとすると，その実行の段階で特権違反という例外処理（8.3節参照）が発生する．

プログラム実行状態の遷移を図8.2にまとめる．ユーザ状態からスーパバイザ状態への遷移は，図8.1に示したプロセッサ処理状態における通常状態から例外状態への遷移が行われるのと同時に行われる．すなわち，例外状態で実行される例外処理はスーパバイザ状態で実行される．例外処理終了後，プロセッサ処理状態が通常状態へと遷移してもステータス レジスタ内のSビットは，まだ“1”のままである．そのため，例外処理後のプログラムは，まずスーパバイザ状態から開始される．プログラム実行状態をスーパバイザ状態からユーザ状態に遷移させるには，スーパバイザ状態でのみ使用が許されている特権命令（図8.2に示す）を用い，ステータス レジスタ内のSビットを“0”に書き換えればよい．

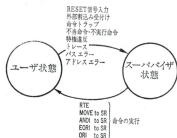



図 8.2 プログラム実行状態の遷移

プログラム実行状態による区別は，特権命令の使用を規定するのみでなく，命令オペランド等の情報の参照にも影響を与える．たとえば68000の内部レジスタであるアドレス レジスタA7は，システム スタック ポインタとしても用いられる．このA7として指定されるアドレス レジスタ（システム スタック ポインタ）は2本存在する．一方はユーザ状態で指定されるユーザ システム スタック ポインタ（USP）であり，もう一方がスーパバイザ状態で指定されるスーパバイザ システム スタック ポインタ（SSP）である．プログラム実行状態によって，どちらのレジスタを使用するのかが自動的に決められる．

メモリ参照の際にも，プログラム実行状態を区別するため，アドレス信号の出力（A1～A23および$\overline{LDS}$, $\overline{UDS}$）と同期してファンクション コード（FC0～2）が出力される．ファンクション コードによる分類はすでに表3.2に示した．FC2がプログラム実行状態を表し，FC0, FC1でプログラム参照，データ参照の区別を表している．この信号を有効に利用することによって，ユーザ状態でのプログラム実行中に，スーパバイザ領域のデータの参照を制限することができる．

8.3 例 外 処 理

8.1および8.2節で説明したように，外部割込み，TRAP命令等により，プロセッサ処理状態は例外状態となり，プログラム実行状態はスーパバイザ状態へと遷移する．本節では，この例外状態におけるプロセッサの動作（以後，例外処理と表記）について説明する．

まず例外処理の概要を，図8.3を用いて説明する．例外処理は5種類の動作（分類番号[1]～[5]，各要因別の説明のためにフローチャート内に分類番号を記載した）に大別される．これらの動作は，例外処理発生要因ごとに多少異なるので，概要説明後に要因別の詳細説明を行う．

例外処理を必要とする要因が発生すると，以下に述べる例外処理が開始される．[1]では，まず例外処理を行う前のステータス レジスタの内容を68000内システム レジスタ(ユーザが直接アクセスすることはできないレジスタ）に退避する．その後，ステータス レジスタのSビット（ビット番号13）を“1”にして，スーパバイザ状態に移行させる．同時に，例外処理終了後に起動されるプログラムでのトレースを抑止するため，Tビット（ビット番号15）を“0”とする．

[2]では例外ベクタ（例外処理終了後に起動されるプログラムのアドレス）を獲得するための準備が行われる．すなわち例外処理を発生させた要因に従って68000内で自動生成させた例外ベクタ番号（たとえば，バス エラーの場合の例外ベクタ番号は2），あるいは外部デバイスが68000に知らせた例外ベクタ番号（101ページ，割込み例外処理参照）を4倍して，例外ベクタのアドレスとする．例外処理発生要因と，例外ベクタ番号および例外ベクタアドレスの関係を表8.1にまとめる．ベクタ番号64～255が，一般の外部デバイスから68000に要因を知らせるために用意されている例外ベクタ番号である．

[3]では，例外処理を行う前の68000内部状態をメモリに退避する．スーパバイザ システム スタック（SSPが指定するアドレスのメモリ上）にプログラム カウンタの内容（この値は，ここまでの例外処理では変化しない）および例外処理を行う前のステータス レジスタの内容（[1]でシステム レジスタに退避した値）を格納する．これらの格納は，最初スーパバイザ スタック ポインタが指定していたアドレスからマイナス2バイトした位置を先頭としてプログラム カウンタの下位2バイト分を，さらにアドレスが減少する方向に，プログラム カウンタの上位2バイト分，システム レジスタに退避されたステータス レジス

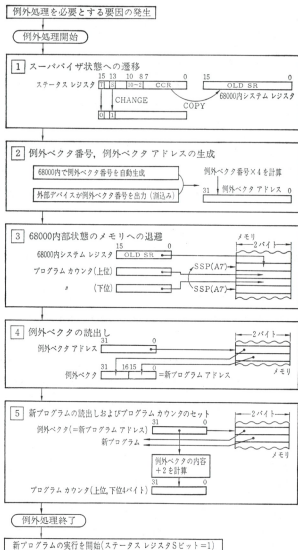



図 8.3 例 外 処 理 の 概 要

タの内容を書き込んでいく．書込みが終了した時点では，スタック ポインタの内容は，書込みを行う前の値から6マイナスした値（バス エラー，アドレス エラーでは14マイナスした値）となっている．

表 8.1　例外ベクタの割当て

Vector Number(s)	Address			Assignment
	Dec	Hex	Space	
0	0	000	SP	Reset: Initial SSP
1	4	004	SP	Reset: Initial PC
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12*	48	030	SD	(Unassigned, Reserved)
13*	52	034	SD	(Unassigned, Reserved)
14*	56	038	SD	(Unassigned, Reserved)
15	60	03C	SD	Uninitialized Interrupt Vector
16〜23*	64 95	04C 05F	SD	(Unassigned, Reserved)
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Auto-vector
26	104	068	SD	Level 2 Interrupt Auto-vector
27	108	06C	SD	Level 3 Interrupt Auto-vector
28	112	070	SD	Level 4 Interrupt Auto-vector
29	116	074	SD	Level 5 Interrupt Auto-vector
30	120	078	SD	Level 6 Interrupt Auto-vector
31	124	07C	SD	Level 7 Interrupt Auto-vector
32〜47	128 191	080 0BF	SD	TRAP Instruction Vectors
48〜63*	192 255	0C0 0FF	SD	(Unassigned, Reserved)
64〜255	256 1023	100 3FF	SD	User Interrupt Vectors

SP: Supervisor Program, SD: Supervisor Data

* ベクタ番号12，13，14，16〜23，48〜63は将来使用する可能性がある．ユーザの周辺LSIにこれの番号を割り当ててはならない．

④では，②で生成した例外ベクタのアドレスを用いて，例外処理終了後に実行する新プログラムのアドレス（例外ベクタ）を読み出す．

そして⑤で，この例外ベクタとして読み出した新プログラムのアドレスを用いて，新プ

表 8.2　例外処理の分類と優先度

グループ	例外処理の種類	例外処理発生要因の検出	例外処理の実行
0	リセット アドレス エラー バス エラー	クロック サイクル毎に検出	次のマイナ サイクル（2クロック サイクル単位）から実行
1	トレース	命令サイクルの最後で検出	次の命令の実行直前に実行
	割込み 不当命令 未実装命令 特権違反	その直前に実行中の命令サイクル内で検出	その命令実行の直前に実行
2	命令トラップ （TRAP, TRAPV, CHK, 除数0）	その命令の命令サイクル内で検出	その命令実行の途中から実行

ログラムを4バイト読み出す．同時に，プログラム カウンタの内容を，例外ベクタで指定されたアドレス値に2を加えた値とする．

以上で例外処理は完了し，⑤で読み出した新プログラム（エラー回復処理を行うシステム プログラム）の実行を開始する．なお，ステータス レジスタのSビットは“1”のままであるから，プログラム実行状態はスーパバイザ状態である．

複数の例外処理発生要因が同時に示された場合には，表8.2に説明する優先度に従って，1つずつ順番に処理される．例外処理発生要因の検出タイミングには3種類あり，1クロック毎に検出するものをグループ0，次の命令を実行する前に検出するものをグループ1，命令実行のシーケンス内で検出するものをグループ2と分類する．表8.2は，この分類に従ってグループ分けした例外処理発生要因である．これらグループ間での例外処理優先度は，グループ0が一番高く，グループ2が一番低い．そのため，たとえばステータス レジスタのTビットが1となっていてトレース例外処理を要求する状態であっても，バス エラーが発生した場合には，まず，グループ0のバス エラー例外処理が実行され，そのエラー処理（例外処理およびそれに続くエラー回復用のシステム プログラムの実行）が終了してから，トレース例外処理を実行することになる（ただし，前述システム プログラムの最後で，バス エラー例外処理を行う前のステータス レジスタの内容を再びプロセッサ内のステータス レジスタに格納しないと，トレース例外処理は発生しない）．

また，グループ内にも優先度がある．グループ0では，リセット例外処理の優先度が最も高く，バス エラー例外処理の優先度が最も低い．そのため，同時にバス エラーとアドレス エラーが発生した場合には，まず，アドレス エラー例外処理が行われる．グループ

1での優先度は，高い方から，トレース例外処理，割込み例外処理，不当命令・未実装命令の例外処理，特権違反例外処理の順である．しかし，グループ2内では優先度という考え方はない．これは，グループ2の例外処理の発生はある特定の命令の実行に起因しており，これらの要因が同時に発生することはないからである．

a．リセット例外処理　外部リセットピン入力のアサートによって発生する例外処理をリセット例外処理という．リセット例外処理は，システムの初期状態設定および重大な故障（2重バス エラー等）からの回復を図るためのものであり，最も優先度の高い例外処理である．これによってシステム全体がリセットされる．一方，特権命令として提供されている RESET 命令は，68000に接続された外部デバイスをリセットするためのものである．そのため，この特権命令を実行すると，外部リセットピン出力はアサートされるが，

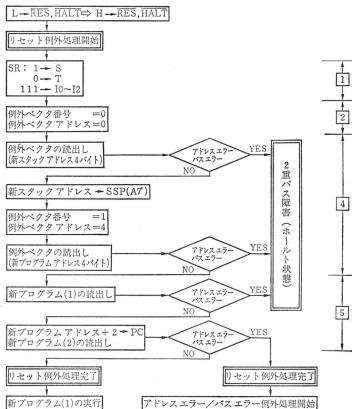



図 8.4 リセット例外処理シーケンス

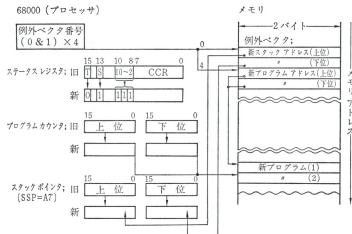



図 8.5 リセット例外処理

例外処理は行われない（17章，特権命令参照）．

リセット例外処理の詳細を，図8.4および図8.5を用いて説明する．図8.4の右側に示した分類番号1～5は，例外処理の概要を説明した図8.3の分類番号1～5に対応している．例外処理が開始されると，まず，スーパバイザ状態への遷移が行われる．リセット例外処理では，例外処理を行う前の68000内部状態を記録しておく必要がないので，ステータス レジスタのシステム レジスタへの一時退避は行われない．

次に，例外ベクタ番号および例外ベクタ アドレスの生成が行われる．

最初に生成される例外ベクタ番号0（例外ベクタ アドレス＝0）からの4バイトにはスーパバイザ スタック ポインタの初期値（スタック アドレス）が格納されている．その新スタック アドレスは，メモリより読み出され，スーパバイザ スタック ポインタ（SSP＝A7'）にセットされる．その後，例外ベクタ番号1（例外ベクタ アドレス＝4）からの4バイトに格納されている新プログラムアドレスがメモリより読み出される．

8バイトの例外ベクタ0番と1番の読出しが終了すると，リセット例外処理終了後に実行される新プログラムの読出しが行われる．この新プログラムの先頭アドレスとして，例外ベクタ1番から読み出された例外ベクタの内容が用いられる．この先頭アドレス（例外ベクタ）からの4バイトが，新プログラムとして読み出される．さらにプログラム カウンタには，例外ベクタから読み出された先頭アドレスに2を加えた値がセットされる．

2重バス障害の発生する可能性があるのは，例外ベクタ読出しおよび新プログラムの最

初の2バイト分の読出しを行うときである．この読出し時に，バス エラーが発生すると2重バス障害とみなされ，プロセッサはホールト状態となる（図8.4参照）．

b. 割込み例外処理　割込み要求の入力ピン $\overline{\mathrm{IPL}}$0〜2に外部割込みレベルをセットすることによって，プロセッサに割込み要求が行われる．外部割込みレベルには7レベルがある．このレベルは1から7まで番号で分類されており，レベル7が最も優先度の高い割込みレベルである．外部割込みレベル0は割込み要求がないことを表す．一方，ステータス レジスタには，現在プロセッサが実行しているプログラムの優先度を表す割込みマスク情報I0〜2がある．この割込みマスク情報で示されたレベルより高いレベルの割込みが受け付けられる．

プロセッサは割込み要求を受け取っても即座に割込み例外処理を実行することはせず，現在実行中の命令が終了するまで，その要求を待機させる．待機させられた割込み要求の外部割込みレベルは割込みマスク情報と比較され，そのレベルがマスク情報で示されるレベル以下の場合には次の命令が継続され，割込み例外処理の実行は延期される．待機させられた割込み要求の外部割込みレベルが割込みマスク情報で示されるレベルより高い場合には，現在実行中の命令終了後，割込み例外処理を開始する．外部割込みレベルおよび割込みマスク情報について表8.3に示す．ここで $\overline{\mathrm{IPL}}$0〜2のピンに入力する外部割込みレベルは負論理，ステータス レジスタのマスク情報I0〜2は正論理であるので注意された

表 8.3　割 込 み レ ベ ル

割込み優先度	レベル	外部割込みレベル（ピン入力する割込みレベル）				割込みマスク情報（ステータス レジスタに設定する内部実行レベル）			
		$\overline{\mathrm{IPL}}$ 2	1	0	要求割込みレベル	I2 (SRビット10)	I1 (ビット9)	I0 (ビット8)	設定マスク情報
高 ↑	7	L	L	L	マスク不可能な割込み要求	1	1	1	レベル7以外の割込み要求をマスクする
	6	L	L	H	マスクされる可能性のある割込み要求	1	1	0	レベル 6〜1 の割込み要求をマスクする
	5	L	H	L		1	0	1	レベル 5〜1 の割込み要求をマスクする
	4	L	H	H		1	0	0	レベル 4〜1 の割込み要求をマスクする
	3	H	L	L		0	1	1	レベル 3〜1 の割込み要求をマスクする
	2	H	L	H		0	1	0	レベル 2〜1 の割込み要求をマスクする
	1	H	H	L		0	0	1	レベル1の割込み要求をマスクする
低	0	H	H	H	割込み要求を起さない	0	0	0	割込み要求をマスクしない

い．以下，割込み例外処理の詳細について説明する．

割込み要求が受け付けられると，図8.6，8.7に示す割込み例外処理が行われる．まず，例外処理の概要で説明したスーパバイザ状態への遷移①が実行される．このとき，外部割込みレベル入力ピンで指定した割込みレベルを，ステータス レジスタ割込みマスク情報I0〜2にセットする．次いで，現状プログラム カウンタの内容から2を引いた値の下位

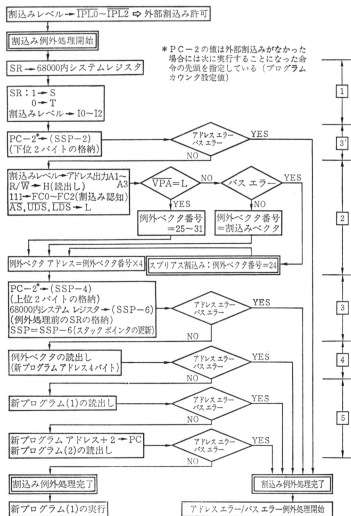



図 8.6　割込み例外処理シーケンス

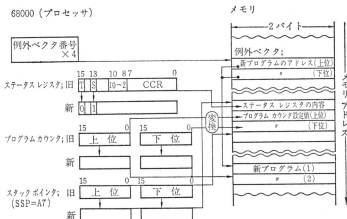



図 8.7　割込みトレース，不当命令，未実装命令，特権違反，命令トラップの例外処理

2バイト分が，スーパバイザ スタック ポインタ（SSP=A7'）によってアドレスを指定されたメモリ上（以後，“スーパバイザ システム スタック上” と表記）に退避される．現状プログラム カウンタの内容から2を引いた値は，例外処理が発生しなかった場合に次に実行する命令の先頭アドレスを表している．

割込み例外処理の特徴は，例外ベクタ番号の発生法である．割込み例外処理のタイミング チャートを図8.8に示すが，例外ベクタ番号の発生は割込みアクノレッジ サイクルで行われる．割込みアクノレッジ サイクルではまず，アドレス出力ピン A1~3 に割込みレベルをセットし，同時にファンクション コード出力ピン FC0~2 に割込み認知を表す “111” をセットして，あたかもメモリの読出しを行うかのように動作する．この読出し動作に対して，割込みを認められた外部デバイスが応答する．この応答で，$\overline{\mathrm{VPA}}$ 入力ピンを “Low” レベルにセットすると，他の例外処理と同様に，例外ベクタ番号は68000内で自動生成される（表8.1 例外ベクタ番号25~31）．一方，$\overline{\mathrm{VPA}}$ 入力ピンを “High” レベルにした場合は，外部デバイスがデータ バスを通して68000に例外ベクタ番号をセットする（68000はデータ バスから読込みを行う）．データ バス上に示される割込みベクタの形式を図8.9に示す．ここで割込みベクタとして指定する例外ベクタ番号は，表8.1で示したように64~255のどれか1つの値とする（ハード的には0~255を指定できるが，すでに例外ベクタ番号として割り当てられているので，割込みベクタ番号として，使用することは好ましくない）．この外部デバイスからの割込みベクタとして指定される例外ベクタ番

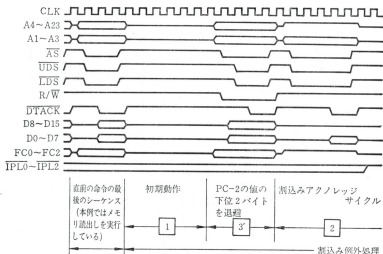



図 8.8　割込み例外処理のシーケンス タイミング

注）割込みベクタとして0～63の例外ベクタ番号を指定することも可能である．しかし，0～63の例外ベクタ番号の使用はある機能を予想して設定されたものであるため，割込みベクタ番号として使用することは好ましくない．

図 8.9　割込み例外処理で外部デバイスが出力する割込みベクタ

号の読込みを68000が失敗した場合（バス エラーピンのアサート，Low→$\overline{\text{BERR}}$）は，スプリアス割込みとして扱われ，例外ベクタ番号24が68000内で自動生成される（この場合のバス エラーは2重バス障害の対象とはならない）．このようにして生成された例外ベクタ番号を4倍して例外ベクタ アドレスを生成するのが，割込みアクノレッジ サイクルに続く2クロックのアイドル期間で行われる．

以下，68000内部状態をメモリのシステム スタック上へ退避（3）し，例外ベクタの読出し（4）を行い，新プログラムの読出しおよびプログラム カウンタのセット（5）が実行される．これで，割込み例外処理は完了し，プログラム実行状態はスーパバイザ状態のままで，新プログラムの実行を開始する．

c.　トレース例外処理　1つの命令を実行する毎にプロセッサの内部状態を記録する手段（トレース機能）があると，プログラムを開発するときには非常に有効である．68000

は，ステータス レジスタのＴビット（ビット番号15）が“1”の場合，1命令の実行が終了する毎にトレース例外処理を発生させる．この例外処理は，トレース用の各種機能をもつシステム プログラムに起動をかけることができる．Ｔビットが“0”の場合には，トレース例外処理は発生せず，次の命令を連続して実行する．

トレース例外処理における68000内レジスタとメモリとのデータのやりとりを図8.7に，シーケンスの詳細を図8.10に示す．トレース例外処理では，1スーパバイザ状態への遷移，2例外ベクタ番号および例外ベクタ アドレスの生成，368000内部状態のメモリへの退避，4例外ベクタの読出し，5新プログラムの読出しおよびプログラム カウンタのセットが順々に実行される．

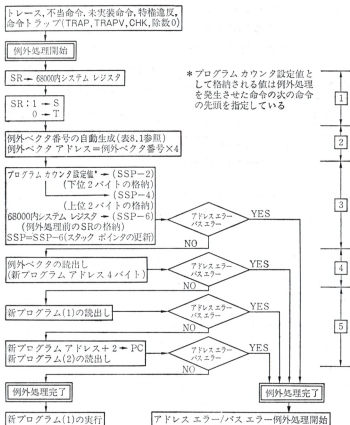



図 8.10　トレース，不当命令，未実装命令，特権違反，命令トラップの例外処理シーケンス

前述の例外処理後に実行する新プログラムで，68000内のレジスタ等をメモリなどに出力しておけば，1命令ごとに68000の動作をモニタすることができ，プログラムを開発する上で非常な助けとなる．なお，このモニタ機能を有する新プログラムは，スーパバイザ状態で実行されるシステム プログラムの一部である．

d．不当命令・未実装命令例外処理　命令として許されていないアドレスモードを指定するなど不当な命令パターンを命令として実行しようとしたときには不当命令例外処理が発生する．不当命令例外処理の詳細を図8.7，8.10に示す．これは，トレース例外処理と全く同じ処理である．

68000ではエミュレーション用の命令パターンとして，図8.11に示す2種類の未実装命令パターンが用意されている．この未実装命令を実行しようとしたときに，未実装命令例外処理が発生する．未実装命令処理の詳細を図8.7, 8.10に示す．例外ベクタ番号は表8.1で示すように，“1010” 未実装命令に対しては10が，“1111” 未実装命令に対しては11が割り当てられている．未実装命令例外処理後に実行するプログラムで，未実装命令のビットパターンを解析し，それに対応して関数演算，浮動小数点演算等の新機能をサポートした後，未実装命令に続く命令を実行するようにすれば，見掛け上はこの未実装命令がその新機能を実行する命令となる（16章参照）．プロセッサの動作としてみた場合には，未実装命令の実行は，ある機能をもったプログラムの起動（呼出し）操作となっている．

図 8.11　不実行命令のオペレーション ワード パターン

e．特権違反例外処理　68000ではシステムの信頼性を向上させるため，プログラム実行状態をスーパバイザ状態とユーザ状態とに区別している（8.2節）．このスーパバイザ状態でだけ利用できる命令として特権命令（詳細に関しては17章，特権命令を参照）がある．特権違反例外処理は，この特権命令をユーザ状態で実行しようとしたときに発生する．特権違反例外処理の詳細を図8.7，8.10に示す．この処理はトレース例外処理等と同様である．

f．命令トラップ例外処理　命令トラップ例外処理は，TRAP命令の実行，およびTRAPV命令，CHK命令，除算命令の実行時にプロセッサの内部状態がある設定条件となった場合に発生する．例外処理は図8.7，8.10に示すように，トレース例外処理等と同様である．

TRAP 命令を実行すると，必ず命令トラップ例外処理が発生する．例外ベクタ番号32～47（表8.1参照）は，命令のオペレーション ワード下位4ビットで指定される．このTRAP命令は，ユーザ状態で実行するプログラム内から，スーパバイザ状態で実行するプログラムを呼び出す場合（スーパバイザ コール）に利用される．

TRAPV 命令を実行すると，前の命令の実行の結果によりコンディション コードのVフラグが“1”となった場合，命令トラップ例外処理が発生する．したがって，オーバフローが発生しそうな命令の直後に TRAPV 命令を置き，例外処理後に起動される新プログラム

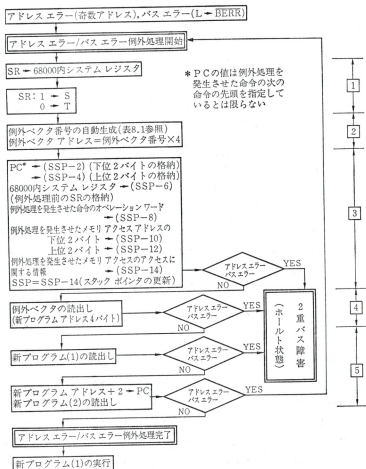



図 8.12　アドレス エラー，バス エラーの例外処理シーケンス

でその対策を行えば，オーバフローによる誤りを防ぐことができる．また，CHK 命令を実行すると，指定したデータ レジスタの内容が 0 以下の場合およびソース オペランドで示した値よりも大きい場合，命令トラップ例外処理が発生する．詳細に関しては，16 章のシステム制御命令を参照されたい．

除算命令を実行し，除数が 0 の場合にも命令トラップ例外処理が発生する．なお，除算結果がオーバフローする場合には例外処理は発生せず，除算命令は実行されない．オーバフローを対策するためには，除算命令の後に TRAPV 命令を置き，例外処理に実行される新プログラムでエラー処理を行えばよい．除算命令に関しての詳細は，13 章の算術論理演算を参照されたい．

g. アドレス エラー例外処理　ワードまたはロング ワードのメモリ参照をする場合，アクセスしたメモリのアドレスが奇数であると，アドレス エラーが発生し例外処理が行われる．アドレス エラー例外処理においても，図 8.12 に示すように，1スーパバイザ状態への遷移，2例外ベクタ番号（=2，表 8.1 参照）および例外ベクタ アドレス（=2×4=8）の生成，が実行される．レジスタとメモリの間のデータの流れを図 8.13 に示す．

アドレス エラー例外処理での特徴は，メモリ上に退避される 68000 内部状態の情報である．システム スタック上には，図 8.13 で示すように，プログラム カウンタの内容，例外処理実行前のステータス レジスタの内容のほかに，例外処理を発生させた実行中の命令のオペレーション ワード，例外処理を発生させたメモリ アクセス アドレス，およびそのアクセスに関する情報が退避される．メモリ上に退避されるプログラム カウンタの内容は，必ずしも命令の先頭アドレスではなく，例外処理を発生させた現在実行中の命令拡張部から次に続く命令，あるいは次に実行する予定の命令の命令拡張部までのどこか 1 つの途中のアドレスを指定している．このように，指定アドレスが一定していないので，例外処理後に実行する新プログラムのなかで，メモリ上に退避された 68000 内部状態の情報を解析して，エラー処理を行う必要がある．

68000 内部状態のメモリへの退避後には，4 例外ベクタの読出し，5新プログラムの読出しおよびプログラム カウンタのセットが行われ，例外処理を完了する．

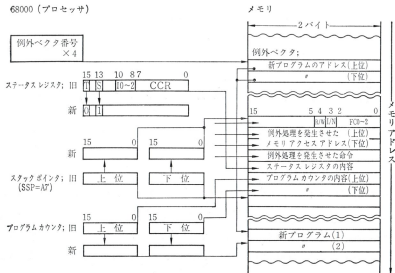
h. バス エラー例外処理　メモリをアクセスしたとき，メモリからの応答がない場合などには，外部回路が 68000 にバス エラーの発生を知らせる必要がある．これは，バス エラー入力ピンをアサートすることにより行われる．68000 がバス エラーの発生を検出すると，バス エラー例外処理が発生する．メモリからの応答がなくても，バス エラー入力ピンのアサートが行われなければ，68000 はメモリからの応答を待ち続けることになる．

注） メモリ上に格納される68000内部情報
R/W: アドレス エラー, バス エラーを発生させたメモリ アクセスの状態（1: 読出し, 0: 書込み）
I/N: アドレス エラー, バス エラーを発生させたときのアクセスされたデータの内容（1: 命令, 0: 命令以外）

図 8.13　アドレス エラー, バス エラー例外処理

バス エラー例外処理の詳細を図8.12, 8.13に示す. この処理はアドレスエ ラー例外処理と同様である. 例外処理後に実行する新プログラムでエラー処理を行う場合には, アドレス エラー例外処理の項で説明したように, メモリ上に退避されたプログラム カウンタの内容値が所定のアドレスを指定していないことに注意する必要がある.

9　周辺デバイスとのインターフェイス

68000 MPU にはメモリ，68000 周辺 LSI の他に，8ビット マイクロプロセッサ 6800 周辺 LSI を接続できる．68000 周辺 LSI については次章で述べることにし，この章では，メモリおよび 6800 周辺 LSI を接続する場合に必要となるインターフェイスについて説明する．68000 MPU と周辺 LSI とを接続する場合，68000 のアーキテクチャ（2章で説明）のうち次の2点が特徴となっている．

（1）　メモリ マップドI/O

（2）　同期および非同期バス インターフェイス

メモリ マップドI/Oは，プログラムおよびデータとなる ROM，RAM のアドレスと入出力装置（I/O）のアドレスを同一アドレス空間に割り付けるものである．これによりバスを介してシステムを拡張することが容易になり，プログラムからはメモリも入出力装置も同じものとして扱うことができる．68000 バス インターフェイスは同期および非同期の両方式に適用可能であり，それぞれに必要な制御信号を有する．なお，同期バス インターフェイスは 6800 周辺 LSI と接続するためのものである．

各々の周辺デバイスとの接続を説明する前に，68000 を用いたシステムの全体構成の例を図 9.1 に示す．図 9.1 は2個の 68000 を用いマルチ プロセッサ システムを構成した例である．68000 システム（I）および システム（II）はバス アービタ，バス結合デバイスを用いてそれぞれのローカル バスを共通バス（グローバル バス）と結合し，主メモリを共有している．さらにシステム（II）は IPC（Intelligent Peripheral Controller）を用いてプライベート バスを設け，メモリ，I/O を拡張している．

68000は，マルチプロセッサ構造を取りやすいような機能をもっていることも特徴の1つである．このように 68000 は大規模なコンピュータ システムをも実現できるが，本章ではこれらシステムを実現する上で基本となる各種周辺 LSI とのインターフェイスの例を

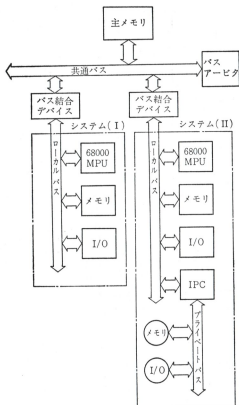



図 9.1 マルチプロセッサ システムの例

いくつか紹介する.

9.1 最小システム構成

68000のシステムを構成する場合には少なくとも次のものが必要である.

(1) 68000 MPU

(2) +5V 電源

(3) TTL レベルの単相クロック

(4) プログラムを書き込んだ ROM (または PROM) および RAM

(5) リセット，ホールト回路

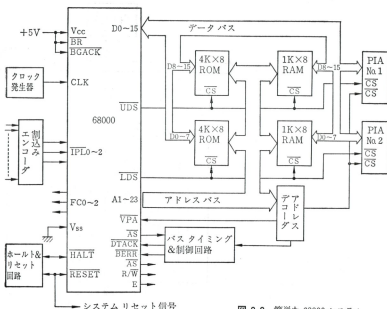



図 9.2　簡単な 68000 システム

（6）　入出力インターフェイス用デバイス

以上の部品を使用して簡単な 68000 システムを構成した例を図 9.2 に示す．この例ではメモリとして 4K×8 ビットの ROM を 2 個使用して 4K ワードを構成し，さらに 1K×8 ビットのスタティック RAM を 2 個実装して 1K ワードを構成している．また入出力用インターフェイスとして PIA を 2 個配置している．この PIA の選択信号は $(\overline{\text{CS}})$ アドレスバスの内容をデコードすることによって得られる．

これらのメモリおよび PIA は $\overline{\text{UDS}}$, $\overline{\text{LDS}}$ によってバイト（8 ビット）単位のアクセスも可能になっている．図では PIA No. 1 はデータワードの下位バイトを，PIA No. 2 はデータワードの上位バイトを構成するように結線されている．

バスタイミング制御回路は $\overline{\text{DTACK}}$ 信号の生成を行う．同時に，外部デバイスが割り付けられていないメモリ空間へのアクセスが生じた場合 $\overline{\text{BERR}}$ 信号をアサートする．

CLK 端子にはデューティ 50% の TTL レベルのクロックを入力する必要があり，クロック発生器はこのためのものである．

割込みエンコーダはプロセッサに対しレベル 1~7 までの割込み要求を入力する．ホールト & リセット回路はリセットならびにホールトオペレーションの制御を行う．パワー

オン リセット時（電源を投入したとき）にプロセッサをリセットするためには 68000 の $\overline{\text{RES}}$ および $\overline{\text{HALT}}$ 端子を 100 ms 以上ローレベルにしなければならない．

$\overline{\text{RES}}$ および $\overline{\text{HALT}}$ 端子はオープン ドレイン端子であり，このため TTL の外部回路ではオープン コレクタ形式の素子を使用しなければならない．また，必要に応じて，3章の図 3.14 で示した簡易形シングル ステップ回路を付加することもある．

9.2 ROM およびスタティック RAM とのインターフェイス

68000 は 23 本のアドレス線 A1～A23 をもっており，2^{23} ワードのアドレスを指定することができる．また，3章で述べたファンクション コード（FC0～FC2）を使用すれば，さらにアドレス空間を4倍に広げることが可能である．上位データ ストローブ（$\overline{\text{UDS}}$）および下位データ ストローブ（$\overline{\text{LDS}}$）によりバイト毎のアクセスもできる．図 9.3 に ROM，スタティック RAM とのインターフェイス例を示す．この例ではすべての

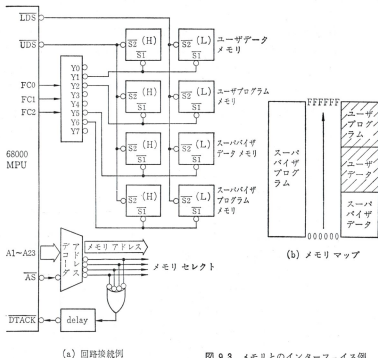



図 9.3 メモリとのインターフェイス例

メモリに FC0〜FC2 を与えることにより，メモリ保護を行っている．また $\overline{\mathrm{UDS}}$, $\overline{\mathrm{LDS}}$ によりバイト アクセスが可能である．プロセッサの 2 クロック サイクル以内でアクセスできない速度の遅いメモリを使用する場合には，メモリ セレクト信号が発生してからメモリが完全に応答するまでの時間を遅延回路を介して $\overline{\mathrm{DTACK}}$ に入力する必要がある．プロセッサの 2 クロック サイクル以内で十分アクセスできる高速のメモリを使用する場合にはこの遅延回路は不要である．

68000 は例外ベクタとしてスーパバイザ プログラム領域の 0〜3 番地およびスーパバイザ データ領域の 4〜255 番地までを予約している．したがってメモリを割り付ける場合にはこの点に注意しなければならない．スーパバイザ データ領域の 256 番地〜1 023 番地までは，ユーザからの割込みによる例外ベクタの領域として使用する．この領域を単なるデータ領域として使うことはさけたほうがよい．

9.3　ダイナミック RAM とのインターフェイス

大容量の RAM をもつシステムを構成する場合は，高集積が可能なダイナミック RAM を使用すると RAM の使用個数が減りコストの点で有利である．ダイナミック RAM は記憶情報を保持するために，適当な周期でリフレッシュ動作を行わなければならない．通常のダイナミック RAM では，この周期は約 2 ms である．

たとえば，日立 HM4816 のような 16K ビットのダイナミック RAM では，128 行×128 列のメモリ構成となっており，アドレス バスの 7 本で行アドレスを与え，別の 7 本で

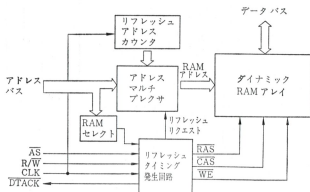



図 9.4　ダイナミック RAM 駆動回路ブロック図

列アドレスを与えることになる．つまり2ms間に行数（128）回だけのリフレッシュ サイクルが必要である．

図9.4にダイナミックRAMの駆動回路のブロック図を示す．リフレッシュ アドレス カウンタはリフレッシュ サイクル中にRAMのリフレッシュ アドレスを発生させるバイナリ カウンタである．上記の16Kビットのダイナミック RAMの場合は7ビットのカウンタが必要である．カウンタ アップの入力はシステム クロックを分周して適当な周期を得る．

アドレス マルチプレクサはリフレッシュ アドレスとシステム アドレス バスの内容を切り換えてRAMのアドレスを発生させるものである．

アドレス マルチプレクサはリフレッシュ サイクル中にはリフレッシュ アドレスを選択する．プロセッサからのメモリ アクセスとリフレッシュ サイクルが競合した場合，リフレッシュ サイクルが優先される．リフレッシュ タイミング発生回路はリフレッシュ サイクルが終了するまでの間 $\overline{\mathrm{DTACK}}$ をアサートしないことにより，プロセッサのメモリアクセスを待たせるわけである．つまり，この場合プロセッサからのメモリ アクセス時間は見掛け上引き延ばされたようになる．

9.4 割込みベクタ発生回路

割込みにはオートベクタ割込みおよびベクタ割込みがある．この両者の識別はVPA入力で行われ，68000が割込み要求を受け付けたことを示す割込みアクノ レッジ サイクルで $\overline{\mathrm{VPA}}$ が外部からアサートされた場合，68000はオートベクタ割込みとみなす．オートベクタ割込みが要求されると，プロセッサは所定の割込みベクタを内部で発生する．一方，外部からベクタ割込みを要求する際にはデータ バスD0～D7を介して例外ベクタ番号を入力する．図9.5に割込み回路の例を示す．この例ではレベル3の割込み（INT3）がオートベクタ割込み，レベル5の割込み（INT5）がベクタ割込みになっている．MPUが割込みを受け付けるとアドレス バスA1～A3から割込みレベルを送出するとともにFC0～FC2をすべて “High” にする．$\overline{\mathrm{VPA}}$ 入力は6800周辺LSIとのインターフェイスのためのコントロール信号にも使用されるため，ワイヤードOR論理が可能なオープン コレクタ形式の素子を外部回路に用いる必要がある．

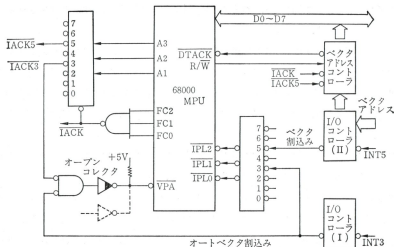



図 9.5　割 込 み 回 路 の 例

9.5　6800 周辺 LSI とのインターフェイス

68000 MPU は 6800 周辺 LSI と直接バス コンパチブルである．6800 周辺 LSI の主なものを以下に示す．

6821　周辺インターフェイス アダプタ (PIA)

6843　フロッピディスク コントローラ (FDC)

6845　CRT コントローラ (CRTC)

6850　非同期コミュニケーション インターフェイス アダプタ (ACIA)

6852　同期シリアル データ アダプタ (SSDA)

これらの 6800 周辺 LSI は同期式デバイスである．68000 とインターフェイスをとるためのブロック図を図 9.6 に示す．アドレス デコーダとアドレス ストローブ信号 (AS) により 6800 周辺 LSI のアドレスが検出されると，68000 MPU は 6800 のバス タイミング条件を満足するようにバス サイクルを修正する．このインターフェイスのために次の 3 つの信号が使用される．

E　(イネーブル)

$\overline{\text{VMA}}$　(バリッド メモリ アドレス)

$\overline{\text{VPA}}$　(バリッド ペリフェラル アドレス)

図 9.6　6800周辺 LSI（デバイス）とのインターフェイス

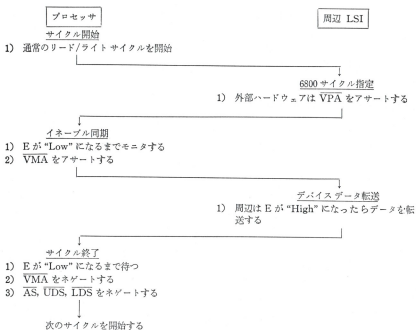
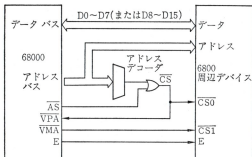



図 9.7　6800周辺 LSI とのインターフェイス フローチャート

図9.7に 6800周辺 LSI とのインターフェイス操作のフローチャートを示す．E 信号は 68000の入力クロック周波数の 1/10の一定周波数をもつクロックである．これは 6800システムにおける E または ϕ_2 に相当するバス クロックである．

$\overline{VPA}$ はアドレスされたデバイスが 6800周辺 LSI であり，データ転送が E 信号と同期して行われるべきであることを 68000 MPU に知らせる．また $\overline{VMA}$ はアドレス バス上に

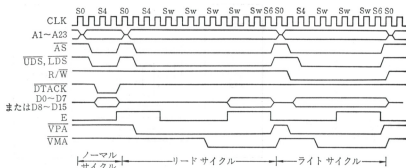



図 9.8　6800 周辺 LSI とのタイミング

有効なアドレスがあり，プロセッサが信号と同期して動作していることを示し，$\overline{\mathrm{VPA}}$ 入力にのみ応答する．$\overline{\mathrm{VMA}}$ 信号は 6800 周辺 LSI のチップセレクト条件の一部として使用される．

以下図 9.8 を用い 6800 周辺 LSI とのインターフェイス タイミングを説明する．

（1）ステート 0（S0）のサイクルのときアドレス バスとファンクション コード（F0～F2）はハイインピーダンス状態となり，半クロック後の S1 でアドレス バスとファンクション コードの出力はハイインピーダンス状態から開放される．

（2）S2 において $\overline{\mathrm{AS}}$ がアサートされ，アドレス バス上に有効アドレスがあることを示す．

バス サイクルがリード サイクルの場合には $\overline{\mathrm{UDS}}$, $\overline{\mathrm{LDS}}$ とも S2 でアサートされる．バス サイクルがライト サイクルの場合には R/$\overline{\mathrm{W}}$ 信号は S2 でライト（“Low”）に切り換り，半クロック後の S3 でライト データをデータ バス上に送り出し，S4 でデータ ストローブを出力する．

$\overline{\mathrm{VPA}}$ 入力信号は $\overline{\mathrm{AS}}$ がアサートされているときに，アドレス バスを外部回路でデコードすることによって作られる．

（3）$\overline{\mathrm{VPA}}$ 検出後，プロセッサは E がネゲート（“Low”）されるまで，必要に応じて待ち状態 Sw に入り，その後 $\overline{\mathrm{VMA}}$ をアサートする．周辺 LSI は E 信号が “High” の間に処理を実行する．

（4）リード サイクルのときには S6 で周辺 LSI からのデータをラッチする．リード サイクル，ライト サイクルのどちらにおいても，プロセッサは S7 で $\overline{\mathrm{AS}}$ とデータ ストローブをネゲートする．信号はこのときに “Low” になる．

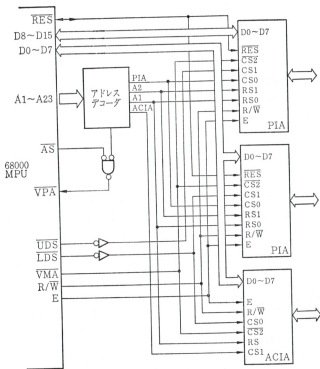



図 9.9 6800 周辺 LSI とのインターフェイス例

（5） 周辺回路は $\overline{AS}$ がネゲートされた後，1クロック以内に $\overline{VPA}$ をネゲートしなければならない．

図9.9に68000と6821 PIA, 6850 ACIA とのインターフェイスの例を示す．2個のPIAはワード（16ビット）毎の入出力とバイト（上位8ビット，または下位8ビット）単位の入出力が可能になっており，ACIAはシステムバスの下位8ビットに接続されている．ここではPIA, ACIAの簡単なインターフェイス例について示した．その他の6800周辺LSIを用いてシステムを拡張することもできる．

10　68000ファミリ周辺LSI

マイクロコンピュータを種々の応用システムに適用するとき，MPU自身の性能とともにその周辺LSIがいかに豊富であるかがそのシステムの性能やシステム開発にとって重要な鍵となる．このため68000でもMPUをサポートする周辺LSIの充実が図られており，現在までに表10.1に示すような周辺LSIが提供されている（開発中のものも含む）．

表 10.1　68000ファミリ周辺LSI

品番号	略　称	名　　称
68450	DMAC*	Direct Memory Access Controller
68451	MMU*	Memory Management Unit
68120	IPC*	Intelligent Peripheral Controller
68230	PI/T*	Parallel Interface/Timer
68122	CTC	Cluster Terminal Controller
68452	BAM	Bus Arbitration Module
68590	LANCE**	Local Area Network Controller
68652	MPCC	Multi-Protocol Communications Controller
68653	PGC	Polynomial Generator Checker
68661	EPCI	Enhanced Programmable Communication Interface
68881	FPCP**	Floating Point Co-Processor

*　本章で説明する，**　開発中

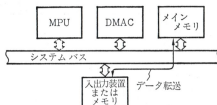
この章では，システムを構成する上で特に重要な68450DMAC，68451MMU，68120IPC，68230PI/Tについて説明を行う．

10.1　68450DMAC (Direct Memory Access Controller)

10.1.1　68450DMACの特徴

68450DMAC (Direct Memery Access Controller) はDMA転送を制御する68000周

図 10.1　DMA 転送

辺 LSI である．DMA 転送とは，図 10.1 に示すとおり，入出力装置とメモリあるいはメモリとメモリとの間のデータ転送を直接（MPU を介さず）行う機能である．図 10.1 において MPU はシステム バスから切り離された状態にあり，DMAC がバス マスタ権をもちデータの転送を制御する．この制御により大量のデータ転送を高速に行うことができる．DMAC はコンピュータ システムでは最も重要な周辺デバイスの 1 つである．

68450 DMAC は 64 ピン パッケージに実装されており以下の機能をもっている．

（1） 68000 MPU および 6800 ファミリとバス コンパチブルである．

（2） 独立した 4 個のチャネルをもつ．

（3） 転送速度は最高 4 M バイト/秒まで可能である．

（4） 論理データ幅は 8，16，32 ビットである．

（5） 物理バス幅は 8，16 ビットである．

（6） 転送リクエスト モードとして次の 3 とおりが用意されている．

i） サイクル スチール モード

ii） バースト モード

iii） オートリクエスト モード（メモリ-メモリ間転送）

（7） 転送アドレッシングとして次の 2 とおりが可能である．

i） シングル アドレッシング モード

ii） デュアル アドレッシング モード

（8） 次の 3 とおりの方法によりマルチブロック転送ができる．

i） 継続動作法

ii） アレイ チェイニング

iii） リンク チェイニング

項目 (6)，(7)，(8) については 10.1.3 項にて説明する．

10.1.2　信号線とバス サイクル

図 10.2 に 68450 DMAC の入出力信号の構成を示す．また，表 10.2 に各信号線の機能をまとめる．

表 10.2　68450 DMAC の信号線

略　号	信　号　名	機　　能
A8～A23/D0～D15	—	アドレス線 A8～A23/データ線 D0～D15
$\overline{CS}$	Chip Select	チップが選択されたことを示す入力信号
A1～A7 $\overline{AS}$, $\overline{LDS}$, $\overline{UDS}$ R/$\overline{W}$, $\overline{DTACK}$, $\overline{BR}$, $\overline{BG}$, $\overline{BGACK}$ V_{CC}, V_{SS}, CLK	—	68000 MPU と同じ機能（3章参照）
$\overline{IRQ}$	Interrupt Request	MPU に対する割込み要求出力
$\overline{IACK}$	Interrupt Acknowledge	MPU が割込みを受け付けたことを示す入力信号
$\overline{OWN}$	Own	DMAC がバス マスタであることを示す
$\overline{UAS}$	Upper Address Strobe	A 8～A 23/D 0～D 15 のうち A 8～A 23 が有効であることを示す
$\overline{HIBYTE}$	High Byte	上位バイトが有効（バイト転送時のみ使用）であることを示す
$\overline{DBEN}$	Data Bus Enable	データ バスが有効である
$\overline{DDIR}$	Data Direction	データ転送の方向を示す
BEC 0,1,2	Bus Exception Controls	BEC 0,1,2= 000　リセット，100　未定義 001　リトライ，101　バス エラー 010　未 定 義，110　バス解放後リトライ 011　ホールト，111　例外条件なし
FC 0, 1, 2	Function Codes	FC 0, 1, 2=111 は未定義，他は 68000 と同じ
$\overline{REQ}$ 0, 1, 2, 3	Requests	DMA 転送要求
$\overline{ACK}$ 0, 1, 2, 3	Acknowledges	DMAC が DMA 転送要求を受け付けたことを示す
$\overline{PCL}$ 0, 1, 2, 3	Peripheral Control Lines	入出力装置制御線（詳細省略）
$\overline{DONE}$	Done	ブロック転送が終了したことを示す
$\overline{DTC}$	Device Transfer Complete	1回のデータ転送が完了したことを示す

図 10.4 は 68450 DMAC の制御の下で，メモリから ACK ピンを有するデバイスへ 16 ビットのデータを転送する際のフローチャートである．これは DMA 転送機能のうちで最も基本的なものである．68000 のワード リード フローチャート（3章，図 3.4 参照）と比較すると次の相違点があるが，基本となる転送手順は同じである．

（1） DMAC が $\overline{ACK}$ のアサートおよびネゲートを行う．

（2） $\overline{ACK}$ でアサートされたデバイスがデータを取り込む．

（3） $\overline{DTC}$ 信号により転送の終了を知らせる．

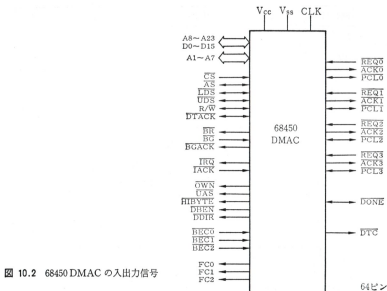



図 10.2 68450 DMAC の入出力信号

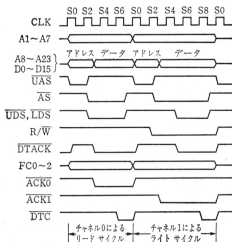



図 10.3 68450 DMAC のリード サイクルおよびライト サイクルのタイミング

図10.3は図10.4で示すDMA転送フローをタイミングチャートで表したものである．リード サイクル（メモリ→I/Oデバイス）は4クロックで，ライト サイクル（I/Oデバイス→メモリ）は5クロックで完了する．68450は独立な4個のチャネルをもっておりバス権を保有したまま別のチャネルのデータ転送を行うこともできる．この際，チャネル切換

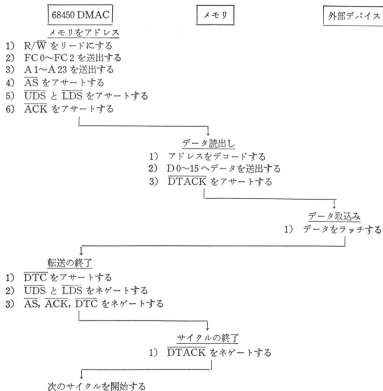



図 10.4　68450 DMAC によるワードリード サイクルのフローチャート

えによるオーバヘッドはない．なお，リード/ライトの定義はメモリを中心にしている．

10.1.3　転送プロトコル

本項では 68450 DMAC の転送プロトコル，すなわち 10.1.1 項であげた転送リクエスト モード，転送アドレッシングおよびマルチブロック転送について述べる．

a．転送リクエスト モード　転送リクエスト モードとは DMA 転送の要求方法のことであり，68450 DMAC では 10.1.1 項でも述べたように，サイクル スチール モード，バースト モード，オートリクエスト モードの 3 とおりがある．

サイクル スチールモードでは，リクエスト信号 $\overline{REQ}$ がアサートされたときに 1 回の DMA データ転送が行われる．すなわち，このモードでは DMA 転送は MPU のバス サイクルの間に 1 バス サイクルだけ割り込んで行われる．したがって，比較的低速のデー

タ転送に用いられる．バースト モードは，$\overline{\text{REQ}}$ 信号がアサートされたときに1ブロックのデータ転送が行われるものである．入出力装置は DMAC からのアクノレッジ信号 $\overline{\text{ACK}}$ によりデータの入出力を行う．1ブロックの転送終了は終了信号 $\overline{\text{DONE}}$ により行われる．このモードでは1ブロックの転送の間，DMAC がバスを占有できるので高速転送に用いられる．オートリクエスト モードは，一種のバースト モードであり内部的に $\overline{\text{REQ}}$ 信号が生成される．メモリは $\overline{\text{REQ}}$ 信号を生成する機能をもたないので，メモリ－メモリ転送はオートリクエスト モードを使う必要がある．転送速度などは DMAC の内部レジスタの値を変えることにより変更できる．以上3種類の転送リクエスト モードをまとめると表 10.3 のとおりとなる．

表 10.3 68450 DMAC における転送リクエスト モードの比較

転送リクエスト モード	転送速度	転送デバイス
サイクル スチール	低 速	入出力装置－メモリ
バースト	高 速	〃
オートリクエスト	高速/低速	メモリ－メモリ

b. 転送アドレッシング　68450 DMAC の転送アドレッシングには 10.1.1 項でも述べたようにシングル/デュアル アドレッシングの2種類がある．これらの相違点は DMAC がアドレス線をとおして選択すべき外部デバイスの個数が1あるいは2個というところにある．シングル アドレッシングでは，DMAC は $\overline{\text{ACK}}$ 信号ならびにアドレス線 A1～A23，バス制御信号で送信デバイスと受信デバイスを選択する．一方，デュアル アドレッシングでは送信デバイスと受信デバイスの選択はアドレス線をとおし2回に分けて行われる．デュアル アドレッシングでは転送すべきデータを一度 DMAC 内部のホールディングレジスタに格納した後，受信デバイスへの書込み操作を行う．したがって，1回の転送でリードおよびライトの両サイクル（合計9クロック）が必要となる．$\overline{\text{ACK}}$ 信号をもたないメモリ－メモリ間転送に対して有効である．また，デュアル アドレッシングでは，2個の8ビット データを 16 ビットデータにパックした後転送するデータ パッキング機能がある．以上の転送アドレッシングについて表 10.4 にまとめる．

表 10.4 68450 DMAC における転送アドレッシング法の比較

転送アドレッシング	クロック数/1転送	転送デバイス	パッキング機能
シ ン グ ル	4クロック（リード） 5クロック（ライト）	入出力装置－メモリ	な し
デ ュ ア ル	9クロック	メモリ－メモリ	あ り

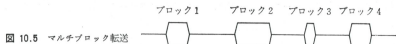



図 10.5 マルチブロック転送

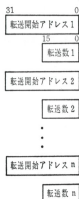



図 10.6 アレイチェイニングによるマルチブロック転送指定フォーマット

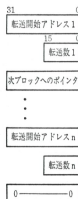



図 10.7 リンクチェイニングによるマルチブロック転送指定フォーマット

c. マルチブロック転送　マルチブロック転送とは図10.5に示すように複数のブロック（一般にブロック長は異なる）を転送するものである．68450 DMACでは10.1.1項でも述べたように，継続動作法，アレイ チェイニング，リンク チェイニングによってマルチブロック転送ができる．継続動作法では原則として転送開始アドレスおよび転送数を毎回設定する必要がある．しかし直前のブロック転送終了時の値がレジスタに保持されているので，同じブロック長でかつ引き続いた番地へデータを転送する場合には継続動作の指令のみをすればよい．アレイ チェイニングにおいては図10.6で示すように転送開始アドレスおよび転送数をメモリ上にアレイ形式で設定する．リンク チェイニングでは図10.7で示すように転送開始アドレスおよび転送数をリスト形式でメモリ上に指定する．

10.1.4 68450 DMACを使用したシステム構成例

図10.8に，68450 DMAC，68000 MPU，メモリ システム(MMUを含む）および68000周辺LSIを用いて，システムを構成した場合の接続参考図を示す．プルアップ抵抗等の詳細については省略してある．

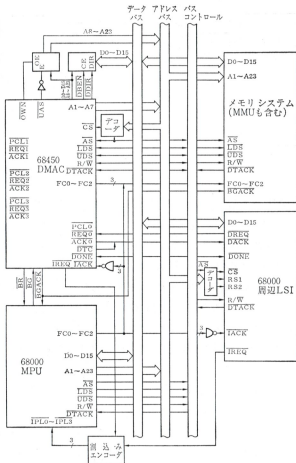



図 10.8 68450 DMAC を使用した構成例

10.2 68451 MMU (Memory Management Unit)

10.2.1 68451 MMU の特徴

68451 MMU はプログラムの論理アドレスを物理アドレスへ変換する LSI である．アドレス変換は論理ブロックごとに行われるが，この論理ブロックをここではセグメントという．図 10.9 に 68451 MMU を用いたシステム ブロック図を示す．68451 MMU には論理アドレス（上位 16 ビット）ファンクション コード，R/$\overline{W}$ 信号が入力され，物理アド

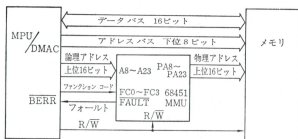



図 10.9　68451 MMUを用いた論理-物理アドレス交換システムのブロック図

レスを出力として生成する．MMUは書込み違反ならびに未定義セグメント（論理アドレスに対応する物理アドレスが未定義）を検出したならば，フォールト信号をバス マスタへ返す．論理アドレス線の下位8ビットはアドレス変換されずセグメント内のアドレスを示す．すなわち1つのセグメントの大きさは最小256バイトである．

このように68451 MMUを用いてシステムを構成すれば，より厳重なメモリ管理を効率良く（ソフトウェアでするよりも）実現でき，OSの負担が軽くなる．しかし，このアドレス変換によって生ずるオーバヘッド（次項で記述）は必ずしも無視できない場合もあり得る．したがって，68451 MMUは小さなシステムよりも大きなシステムに適したものであるといえる．

68451 MMUは次の特徴をもつ．

（1）　68000 MPUおよび68450 DMACとコンパチブル

（2）　24ビット アドレス バスのうち上位16ビット（A8～A23）をアドレス変換する．

（3）　書込みチェックを行う．

（4）　1つのMMUで一度に32個までのセグメントに対しアドレス変換ができる．

（5）　マルチMMUシステムへ容易に拡張できる．

（6）　厳重なメモリ管理が実現でき，OSの負担を軽減できる．

10.2.2　信号線とアドレス交換のバス サイクル

図10.10に68451 MMUの入力出信号の構成を示す．64ピンのパッケージに実装されている．表10.5に各信号線の機能をまとめる．

図10.11にアドレス変換時のバス サイクル タイミング チャートを示す．基本的には68000 MPUのリード/ライト サイクルと同じである．相違点は，MMUが$\overline{\text{AS}}$信号を受

図 10.10　68451 MMU の入出力信号

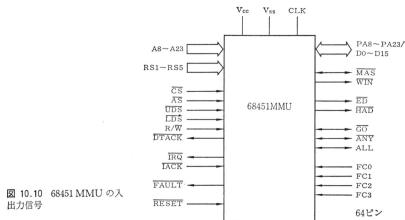



表 10.5　68451 MMU の信号線

略　号	信号名	機　　能
A 8～A 23, $\overline{\text{AS}}$, $\overline{\text{UDS}}$, $\overline{\text{LDS}}$, R/$\overline{\text{W}}$, $\overline{\text{DTACK}}$, Vcc, GND, CLOCK, $\overline{\text{RESET}}$	―	68000 MPU と同じ機能（3章参照）
$\overline{\text{CS}}$, $\overline{\text{IRQ}}$, $\overline{\text{IACK}}$, FC 0, 1, 2	―	68450 DMAC と同じ機能（前節参照）
RS 1～RS 5	Register Selects	内部レジスタ選択線
$\overline{\text{FAULT}}$	Fault	書込み違反，未定義セグメントを検出したことを示す
PA 8～PA 23/D 0～D 15	Physical Address/Data	物理アドレス線/データ線
$\overline{\text{MAS}}$	Mapped Address Strobe	物理アドレス線が有効であることを示す
$\overline{\text{WIN}}$	Write Inhibit	書込み禁止を表す(リード モディファイ ライト時に使用)
$\overline{\text{ED}}$	Enable Data	PA 8～PA 23/D 0～D 15 のデマルチプレクスに使用する
$\overline{\text{HAD}}$	Hold Address	
$\overline{\text{GO}}$, $\overline{\text{ANY}}$, ALL	Global Operation Any, All	マルチ MMU システムを構成するときに使用する
FC 3	Function Code 3	$\overline{\text{BGACK}}$ 信号を入力する

け取ってからアドレス変換を開始し，物理アドレス ストローブ信号 $\overline{\text{MAS}}$ でメモリのアクセスが行われることである．図中“Sw”は 68000 MPU の待ちサイクルを表し，リードサイクルで2クロック，ライト サイクルで1クロックがアドレス変換のオーバヘッドとな

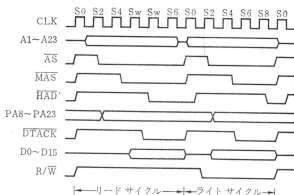



図 10.11　68451 MMU を用いた 68000 システムのリードサイクルおよびライトサイクルのタイミング

る．ライトサイクルにおいて待ちサイクルがない理由は次のとおりである．図 10.11 に示すように，リードサイクルでは 68000 MPU は $\overline{\mathrm{MAS}}$ 信号が確定した後メモリからデータを読み出すのに対し，ライトサイクルでは $\overline{\mathrm{AS}}$ 信号受信と同時にメモリデータを送出できるからである．

10.2.3　アドレス変換

本項では 68451 MMU におけるアドレス変換の方式について述べる．68451 MMU では図 10.12 で示すように"デスクリプタ"とよばれるレジスタブロックが 32 個用意されている．デスクリプタは 6 個のレジスタ(計 9 バイト)からなり，アドレス変換機能の中心となるものである．デスクリプタ中の各レジスタの機能についてはアドレス変換過程の説明において適宜ふれる．68451 MMU のアドレス変換は次の 3 段階に分けることができる．

（1） デスクリプタの選出

（2） 書込みチェック

（3） 物理アドレスの生成

以下に各段階の機能について述べる．

図 10.12　68451 MMU のアドレス交換用デスクリプタ

a．デスクリプタの選出　アドレス変換を行うにあたり，68451 MMU は 32 個のデ

スクリプタのうちどれを使うのか決定する必要がある．このために4本のファンクションコード（FC0～FC3）および論理アドレス線（A8～A23）を入力として使用するFC3には $\overline{\text{BGACK}}$ を入力する．まずファンクションコードから内部的にアドレススペース番号という情報を生成し，同じアドレススペース番号をもつデスクリプタを選び出す．この選出は連想記憶方式によって行われる．該当するデスクリプタが存在しない場合 $\overline{\text{FAULT}}$ 信号を生成し，アドレス変換を中断する．次に，選択したデスクリプタの論理ベースアドレスと入力アドレス線が一致するかどうか判定する．一致しない場合は $\overline{\text{FAULT}}$ 信号を生成しアドレス変換を中断する．

アドレススペース番号の一致判定および論理アドレスの一致判定においては，それぞれマスク用レジスタ（アドレススペースマスク，論理アドレスマスク）が用意されている．後者を例にとり図10.13を用いマスク機能について説明する．図で示すように論理アドレスマスクの内容が“0”であるビット位置が一致判定の対象外となる．たとえばA8～A11の4ビットがマスクされた場合，これらのアドレス線は一致判定の対象外となるため実質的にセグメントの大きさが2の11乗バイト，すなわち2048バイトまで広がったことになる．マスクの効果についてはアドレススペース番号の一致判定も同じである．

図 10.13 論理アドレス（入力）と論理ベースアドレス(デスクリプタ)との一致判定

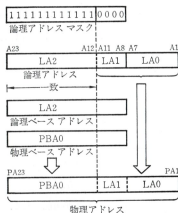



図 10.14 物理アドレスの生成例

b. 書込みチェック　デスクリプタのセグメント状態レジスタにはそのセグメントが書込み可能かどう示すビットがある．書込み禁止のセグメントに対し，R/$\overline{\text{W}}$ 信号が書き込みを示すとき $\overline{\text{FAULT}}$ 信号を生成する．このほかセグメント状態レジスタには，セグメントに対するアクセスの有無，書換えの有無などの情報を示すビットがある．これらはセグメントのスワッピングをはじめとするOS機能のサポートに有用である．

c. 物理アドレス生成　前記（1），（2）の処理を受けたセグメントに対して物理アド

レスが与えられる．図10.14に物理アドレスの生成例を示す．アドレス線A1～A7ならびに論理アドレス マスクでマスクされたアドレス線はそのまま物理アドレスとなる．マスクされなかった部分は物理ベース アドレスで置き換えられる．

10.3 68120 IPC (Intelligent Peripheral Controller)

68120 IPCは入出力装置を制御する汎用のプロセッサであり，MPUと被制御入出力装置との間に接続される．68120 IPCを用いたシステム構成例を図10.15に示す．システムバスにはMPU, DMAC, メイン メモリ，IPCが接続されており，ローカルバスにはローカル メモリ，FDC (Floppy Disk Controller), CRTC (CRT Controller) が接続されている．また IPCはランプ，ブザー等簡単な入出力装置を直接制御することができる．図10.15において68120 IPCの内蔵ROMあるいはローカル メモリにプログラムを格納することにより68120 IPCにインテリジェンスをもたせることができる．したがって，MPUは入出力機能のかなりの部分をIPCに肩代りさせることができ，それだけMPUの負荷が軽減されシステム パフォーマンスの向上につながる．

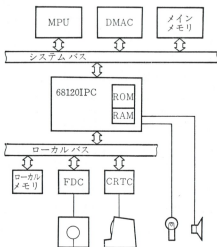



図 10.15　68120 IPCを用いたシステム構成例

68120 IPCは以下の特徴をもつ．

（1） 68000 MPUおよび6800ファミリとバス コンパチブルである．

（2） 8ビット プロセッサ6801を核にしたアーキテクチャをもつ．

（3） 2048バイトのROMを内蔵している．

（4） 128バイトのデュアル ポート RAMを内蔵している．

（5） タスク同期用レジスタ（セマフォア レジスタ）を6個もっている．

（6） 内蔵のROM, RAMの拡張に関して次の3とおりの動作モードが可能である．

　　i） シングルチップ モード

ii) 拡張形非マルチプレックス モード

iii) 拡張形マルチプレックス モード

(7) タイマ機能を有する

(8) シリアル通信機能を有する．

(9) ROM を内蔵しない 68121 というバージョンもある．

図 10.16 に 68120 IPC の入出力信号の構成を示す．48 ピンのパッケージに実装されている．図 10.16 において図中左側のシステム アドレス線から $\overline{\mathrm{RESET}}$ までは 68000 インターフェイス用である．一方，図中右側の信号線は 68120 IPC が制御する入出力装置とのインターフェイス用である．このうちポート 3，SC1，SC2，ポート 4 は 68120 IPC の動作モードにより機能が変化する．これらの機能の相違を表 10.6 に示す．表 10.6 には各動作モードにおける入出力ポートの総数ならびにローカル メモリ空間の大きさも示してある．シングルチップ モードでは計 21 本の入出力ポートが利用でき，拡張非マルチプレックス，拡張マルチプレックス モードではローカル メモリ空間上に入出力コントローラあ

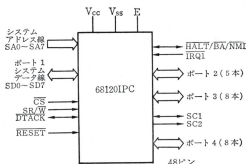



図 10.16 68120 IPC の入出力信号

表 10.6 68120 IPC の動作モードと信号線の機能

信号線 ＼ 動作モード	シングルチップ	拡張非マルチプレックス	拡張マルチプレックス
ポート 2 (5 本)	入出力ポート (パラレル ポート，シリアル ポート，タイマ)		
ポート 3 (8 本)	入出力ポート	D0～D7	A0/D0～A7/D7
SC1 (1 本)	$\overline{\mathrm{OS3}}$*1	R/$\overline{\mathrm{W}}$	R/$\overline{\mathrm{W}}$
SC2 (1 本)	$\overline{\mathrm{IS3}}$*2	$\overline{\mathrm{IOS}}$*3	AS
ポート 4 (8 本)	入出力ポート	A0～A7	A8～A15
入出力ポート総数(含ポート 2)	21 本	5 本	5 本
ローカル メモリ空間	—	128 バイト	64K バイト

*1 Output Strobe, *2 Input Strobe, *3 I/O Strobe

るいはローカル メモリの割当てができる.

ポート2は入出力ポートとしても使えるが，内部レジスタを書き換えることによりタイマ/シリアル通信インターフェイスとしての機能をもつ．タイマ機能として次の3つの条件のときに内部割込みを生成し得る．各々の割込みはマスクをすることもできる.

（1） フリーラン カウンタ（16ビット）が，内部レジスタに設定した数値と一致した場合

（2） フリーラン カウンタがオーバフローした場合

（3） ポート2の最下位ビット（ビット0）のレベル変化（0→1，1→0，指定可能）を検出した場合

シリアル通信インターフェイスとしては，1個の全2重（full-duplex）形非同期チャネルをもっている．転送レートおよび信号形式（Non Return to Zero または Bi-phase）はプログラム可能である．68120 IPC には，パケット通信をサポートするためにウェイクアップという機能がある．これは，MPU が不要パケット（アドレスが一致しない）であると判断した時点で IPC に対し不要パケットの転送を停止させ，そして次のパケットから再び IPC がデータ転送を行うものである．この機能を使うと，MPU はパケットのアドレス部を調べデータ部の取捨を決めればよく，不要なデータ転送がなくなる.

68120 IPC は128バイトのデュアル ポート RAM を内蔵している. この RAM は 68120 IPC の内部プロセッサおよびメイン システムの両方から直接アクセスすることができ，IPC とメインシステムとのデータ転送に使う．しかし，IPC の内部プロセッサとメインシステムが同時に，同じアドレスをアクセスすることは避けなければならない．このため，68120 IPC には6個のセマフォア レジスタが用意されている．図 10.17 にセマフォア レジスタの構成を示す．ビット7の SEM（Semaphore）ビットはテスト アンド セットのビット† である．ビット6の OWN（Ownership）ビットは，SEM ビットをセットしたプロセッサが IPC 自身かメイン システムであるかを示すためのものである.

7	6	5	4	3	2	1	0
SEM	OWN	0	0	0	0	0	0

図 10.17 68120 IPC のセマフォア レジスタ

† 3章および13章参照

10.4 68230 PI/T (Parallel Interface/Timer)

68230 PI/T は入出力ポート，ハンドシェーク機能に加えタイマ機能を有する LSI であり入出力装置の制御用として使う．68230 PI/T は以下の特徴をもつ．

（1） 68000 とバス コンパチブルである．

（2） 最大 24 本までの入出力ポートをもつ．

（3） 24 ビット カウンタと 5 ビット プリスケーラを内蔵し計時タイマとして使用できる．

（4） 図 10.18 で示すように次の 3 とおりのモードで外部へタイミング信号を発生することができる．

i） パルス波形モード（同期，デューティ可変）

ii） 矩形波形モード（同期のみ可変）

iii） ワンショット モード

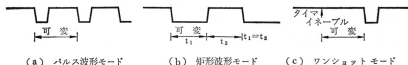



図 10.18 68230 PI/T におけるタイミング発生機能

図 10.19 に 68230 PI/T の入出力信号の構成を示す．48 ピンのパッケージに実装されている．RS1～RS5（Register Select）は PI/T の内部レジスタを選択するための信号線で

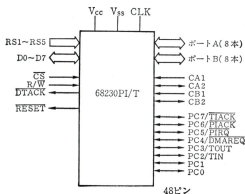



図 10.19 68230 PI/T の入出力信号

ある．データバスD0〜D7が8ビット幅なので68000 MPUのMOVEP命令を使うのが適している．ポートA，ポートBはそれぞれ8ビット幅であり内部は2重バッファ構成となっている．これらのポートをまとめて16ビット幅のポートとして使うことも可能である．CA1, CA2（Strobe Control）はポートAのハンドシェーク用信号線であり，ラッチのタイミングを指定したり，バッファの空き状態の表示などに使われる．CB1, CB2はポートBのハンドシェーク用信号線またはポートA, Bを16ビット幅として使うときのハンドシェーク用信号線である．

PC0〜PC7は8ビット幅のポートとして使うこともできるがPC2〜PC7は内部レジスタを書き換えれば割込み制御，タイマ制御などにも使用できる．$\overline{\text{PIRQ}}$（Port Interrupt Request），$\overline{\text{PIACK}}$（Port Interrupt Acknowledge）はポート割込み制御用の信号である．$\overline{\text{TIACK}}$（Timer Interrupt Acknowledge），TOUT（Timer Out），TIN（Timer In）は68230 PI/Tのタイマ機能に関するものである．68230 PI/Tは前述したとおり24ビットのカウンタ，5ビットのプリスケーラを内蔵している．基本クロックとして内部クロックまたはTINからの外部クロックを使用できる．TOUTからは図10.18で示した3つのモードで外部へ信号を出力できる．TOUTをシステムの割込み要求線へ接続し，$\overline{\text{TIACK}}$により割込みベクトルの発生が可能となる．

68230 PI/Tのタイマ機能はOSが計時用として使用したり，タスクの切換え用として使用するのに適している．

Ⅱ　68000のソフトウェア

11 アセンブラ

この章では68000システムの典型的なアセンブラ[†]の使用法，このうち特にソース プログラムの記述法について説明する．

アセンブラは，ユーザが記述したソース プログラムを機械語に変換するための言語処理プログラムである．ユーザはソースプログラムを，アセンブリ言語すなわち機械命令や制御命令などを表すニモニック記号を用いて記述する．このソース プログラムは図11.1に示すようにアセンブラの入力となる．アセンブラで処理されたプログラムはオブジェクトモジュールとなる．これは仮のアドレス付けをされた機械語の形式になっている．オブジェクト モジュールはリンケージ エディタとよばれる結合編集プログラムによって編集され，実際のアドレスを割り当てられる．こうしてプロセッサが実行可能なロード モジュールが作り出される．

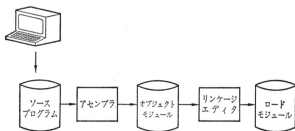



図 11.1 ソース プログラムの変換

† 一般にアセンブラは使用するシステムによって異なるのがふつうである．本書では最も標準的な68000アセンブラを想定している．

11.1 ソース プログラムの構造

ソース プログラムは1つ以上のステートメントによって構成される．ステートメントとは，アセンブリ言語で書かれた1行の文のことで，1行の有効な範囲は1カラムから72カラムまでである．

ステートメントには実行命令，アセンブラ制御命令，コメント文の3種類がある．

実行命令は機械語と1対1に対応するステートメントであり，ソース プログラムの中心をなすものである．アセンブラにより2～10バイトの機械語に変換される．

アセンブラ制御命令はアセンブラの処理を制御するステートメントで，機械語には変換されない．メモリ領域の確保，データの設定，値の割当て，領域の宣言などを行う．

コメント文はプログラム中に注釈を入れるためのものであり，第1カラムが＊で始まる．

11.2 アセンブラの書式

3種類のステートメントのうち，実行命令とアセンブラ制御命令を総称したものを命令ステートメントとよぶことにする．命令ステートメントは図11.2に示す4つのフィールド

	ラベル フィールド	演 算 フィールド	オペランド フィールド	コメント フィールド
例	LABEL 1	ADD. B MOVE. L RTS	(A0), D1 D1, D2	COMMENT HERE COPY (DATA REGISTER) RETURN

図 11.2 命令ステートメントの構成

から構成される．68000アセンブラの記述形式はフリーフォーマットであり，各フィールド間は1つ以上の空白により区別される．次に各フィールドに関する機能と規則を説明する．

11.2.1 ラベル フィールド

ラベルはジャンプ先番地，データ領域番地などを指定するための記号名称であり，ラベル フィールドで定義され，オペランドによって参照される．ラベルは実行命令すべてにつけることができるが，アセンブラ制御命令にはラベルをつけることができない命令がある．ラベルは第1カラムから書くことになっている．また省略する場合には第1カラムを空白

にしなくてはならない.

11.2.2 演算フィールド

演算フィールドでは実行命令やアセンブラ制御命令のニモニックを記入する. 実行命令は演算フィールドのニモニックとオペランドフィールドの記述に基づいて2~10バイトの機械語に変換される. データ定義命令 (DC: Define Constant) ではオペランドの値がメモリにセットされる.

実行命令のニモニックにはアドレス形式により変化形をもつものがある. 変化形は命令のニモニックにA (アドレス), I (イミディエイト), Q (クイック), M (メモリ), X (エクステンド) をつけることによって作ることができる. このうちMとXの場合は必ずつけなければならないが, A, I, Qについては基本形を用いても, アセンブラが自動的に適切な変化形を選択する. しかしプログラムのミスを少なくするためには変化形まで明記しておくことが望ましい†. また, たとえば

```
ADD    #5, D1
```

のようにイミディエイト データが小さい場合, 指定がないとアセンブラは自動的に "ADDQ" を選択するので注意が必要である. 実行命令の基本形と変化形の関係を表11.1に示す.

表 11.1 命令の変化形

命令の基本形	変化形	
	自動的に変化	明記が必要
ADD	ADDA, ADDI, ADDQ	ADDX
AND	ANDI	—
CMP	CMPA, CMPI	CMPM
EOR	EORI	—
MOVE	MOVEA, MOVEQ	—
NEG	—	NEGX
OR	ORI	—
SUB	SUBA, SUBI, SUBQ	SUBX
ROL, ROR	—	ROXL, ROXR

実行命令に対しては扱うデータ サイズが, バイト (8ビット), ワード (16ビット), ロング ワード (32ビット) のいずれなのかを指定する必要がある. これらは命令のニモニックのあとに続けて .B (バイト), .W (ワード), .L (ロング ワード) を書くことによ

† 本書のプログラム例はすべて変化形まで明記した.

って指定する．データ サイズ指定は省略可能であり，その場合には原則としてワードが選ばれる．ただし，命令によっては暗黙的にバイト，ロングワードを選ぶものもあるので，各命令の項あるいは付録を参照されたい．

11.2.3 オペランド フィールド

オペランドフィールドには，命令で定められた個数のオペランドを，定められた順番で記入する．オペランドが2つある場合にはコンマ“，”で区切る．空白を途中に入れると以後の記述がコメントとみなされるので注意する必要がある．またオペランドが2つある場合には第1オペランドがソースオペランドを，第2オペランドがデスティネーションオペランドを表す．図11.3には68000の代表的な命令，ADD, SUB, MOVE 命令について，オペランドの記述方法と演算のしかたを対応させて示した．

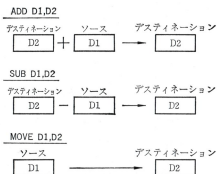



図 11.3　2つのオペランド演算の対応

11.2.4 コメント フィールド

コメント フィールドはステートメントに注釈をつけたいときに使用するものであり省略できる．コメント フィールドには英数字のほかに空白を含め特殊文字が使用できる．

11.3 データ形式

アセンブラで使えるデータ形式には，数値定数，文字定数，記号，式の4種類がある．これらの記述の形式を次に述べる．

11.3.1 数値定数

数値定数は0～9の数字，および A～F の文字を組み合せて得られる値である．以下に示すとおりの数値定数が記述できる．

a. 10進定数　10進定数は0～9までの数字を組み合せて構成される．数値の前に

“+” または “−” の符号をつけることにより正または負の値を表現できる．符号がない場合には正の値を表す．

135	10進で135を表す
+348	10進で348を表す
−79	10進で −79を表す

b. 16進定数　16進定数は0〜9の数字およびA〜Fの文字を組み合せて作られ，先頭に “$” をつけて区別する．なお日本国内では “¥” を用いる場合もある．16進数に符号をつけることはできない．

$10	10進の16に等しい
$A000	10進の40960に等しい

c. 2進定数　2進定数は0,1の数字で構成され，先頭に “%” をつけて区別する．2進数に符号をつけることはできない．

%10011101	10進の157に等しい
%00100101	10進の37に等しい

11.3.2 文字定数

文字定数は英数字および特殊文字のほとんどが使用できる．文字定数の指定は '（アポストロフィ）で囲むことにより行う．文字定数は次の例のように表す．

'AB8X'	文字列　AB8Xを表す
'#%, SP'	文字列　#%, SPを表す
'AP''&+'	文字列　AP'&+ を表す

11.3.3 記号

記号は，英字 [A]〜[Z]，数字 [0]〜[9]，記号 [@]，[$]，[.]（ピリオド）からなる文字列であり，先頭は英字または記号 [@]，[.]（ピリオド）のいずれかでなければならない．文字列の最大長はアセンブラの種類によって異なるが，8字から30字程度である．記号のうち

A0, A1, …, A7, D0, D1, …, D7, CCR, SR, SP, USP

はアセンブラが予約語として使用するため，ラベルとして使うことはできない．なおアセンブラによっては別の予約語を使うものもある．記号の値は，ラベル フィールドに現れた

とき定義される．記号はラベル フィールドに2回以上現れてはならない．

記号に与えられた値は次のように決る．

a. 実行命令および制御命令（EQUを除く）のラベルとして現れたとき　このときのロケーション カウンタ（アセンブラが仮に決めるプログラム カウンタ）の値が記号に与えられる．この値は一般には相対値であるため，このような記号を相対値記号とよぶ†1．

b. EQU命令のラベルとして現れたとき　この命令のオペランド部の値が与えられる．記号の特性は，オペランドが数値定数あるいはそれを演算したものであれば絶対値記号（値が確定）になり，相対値記号をオペランドとしていれば相対値記号となる．

これまで述べた以外に特別な記号として＊（アステリスク）がある．これは＊が現れているステートメントの先頭のロケーション カウンタ†2の値を示すものであり，相対値記号として扱われるラベルフィールドに使うことはできない．

11.3.4 式

オペランド フィールドには式を書くことができる．アセンブラは式の値を計算しその結果をオペランドとする．演算子としては＋と－が許されているが，乗算/除算まで許すものもある．

式には数値定数および記号を使うことができる．記号には前に述べたように絶対値と相対値記号があり，同様に式にも絶対値式と相対値式の区別がある．絶対値式は数値定数および絶対値記号だけを含む式であり，相対値式は絶対値式と相対値記号を含む式である．また相対値記号の和は許されないが，差は絶対値式として扱われる．次に式の例を示す．ここでABS 1，ABS 2はEQUで定義された絶対値記号であり，REL 1は実行命令のラベルにつけられた相対値記号であるとする．

［絶対値式の例］

1234

ABS 1＋ABS 2－100

［相対値式の例］

REL 1－ABS 1＋200

＊＋ABS 1

---

†1 これは11.6節で述べるリロケータブルなプログラミングに関連しており，リンケージのときに絶対アドレスが確定する．

†2 ロケーション カウンタはアセンブラ プログラムの中で定義された仮のプログラム カウンタである．

11.4 実効アドレスとアドレス形式

68000で利用できるアドレス形式は14種類あるが，これをアセンブラで記述するという立場からみると13種類に分類することができる．機械語レベルの分類との対応を表11.2に示す．アセンブラ レベルの分類のうち1～12番目までがオペランドフィールドにアドレス形式を明記するものである．一方13番目のインプライド形式は，演算フィールドで自動的にオペランドが決るものあるいはオペランドのないものであって，オペランド フィールドには何も書かない．この形式をとる命令には，RTE, RTS, RTR, RESET, TRAPV, NOPがある．次にオペランドを明記するアドレス形式の記述のしかたを述べる．

表 11.2 アドレス形式の分類

No.	機械語レベルの分類
1	データ レジスタ直接形式
2	アドレス レジスタ直接形式
3	アドレス レジスタ間接形式
4	ポスト インクリメント アドレス レジスタ間接形式
5	プリデクリメント アドレス レジスタ間接形式
6	ディスプレースメント付 アドレス レジスタ間接形式
7	インデックス付 アドレス レジスタ間接形式
8	長絶対アドレス形式
9	短絶対アドレス形式
10	ディスプレースメント付 プログラム カウンタ相対形式
11	インデックス付 プログラム カウンタ相対形式
12	イミディエイト データ形式
13	クイック イミディエイト データ形式
14	インプライド形式

No.	アセンブラ レベルの分類
1	データ レジスタ直接形式
2	アドレス レジスタ直接形式
3	アドレス レジスタ間接形式
4	ポスト インクリメント アドレス レジスタ間接形式
5	プリデクリメント アドレス レジスタ間接形式
6	ディスプレースメント付 アドレス レジスタ間接形式
7	インデックス付 アドレス レジスタ間接形式
8	絶対アドレス形式
9	ディスプレースメント付 プログラム カウンタ相対形式
10	インデックス付 プログラム カウンタ相対形式
11	イミディエイト データ形式
12	CCR/SR 形式
13	インプライド形式

11.4.1 データ レジスタ直接形式

オペランド フィールドで指定したデータ レジスタが実効アドレスとなる． アセンブラ表現は Dn (n=0~7) である．下線は当該アドレス形式を表す (以下同様).

[例]	ADD .W	A1, D2	[D2+A1→D2]
	MOVE .B	D0, D5	[D0→D5]

11.4.2 アドレス レジスタ直接形式

指定したアドレス レジスタが実効アドレスとなる． アセンブラ表現は An (n=0~7) あるいは SP である．SP は A7 または A7′ と同じであり，システム スタック ポインタを指す．ユーザ システム スタックポインタが使われるか， スーパバイザ システム スタック ポインタが使われるかは実行時のプロセッサのプログラム実行状態によって決る．また，この形式ではバイトサイズのデータを扱うことはできない．

なお特殊な命令として， スーパバイザ状態の中でユーザ スタックを操作するための MOVE 命令がある．この場合だけはオペランドに USP と書きユーザ スタック ポインタを指すことができる．

[例]	SUBA .W	A1, A2	[A2−A1→A2]
	MOVE .L	USP, A1	[USP→A1]

11.4.3 アドレス レジスタ間接形式

アドレス レジスタの内容が指すメモリのデータが実効アドレスとなる． 記述形式は (An) である (n=0~7).

[例]	MOVE .B	(A1), D2	[(A1)→D2]
	TST .L	(A1)	[(A1)−0]

11.4.4 ポスト インクリメント アドレスレジスタ間接形式

アドレス レジスタの内容が指すメモリのデータが実行の対象となる．実行後アドレス レジスタには， 扱ったデータ サイズに従って 1 (バイト)，2 (ワード)，4 (ロング ワード) が加算される．記述形式は (An)+ である (n=0~7).

[例]	ADD .B	(A3)+, D5	[(D5)+A3→D5, A3+1→A3]

11.4.5 プリデクリメント アドレスレジスタ間接形式

指定したアドレス レジスタから，あらかじめ扱うデータ サイズに従って，1，2，または4を引いたあと，アドレス レジスタ間接形式の処理をする．記述形式は −(An) である（n=0〜7）．

```
[例]  MOVE .W    −(A4), −(A5)
      [A4−2→A4, A5−2→A5, (A4)→(A5)]
```

11.4.6 ディスプレースメント付アドレス レジスタ間接形式

指定したアドレス レジスタと命令の中のディスプレースメントを加えた値が指すメモリの内容が実効アドレスとなる．ディスプレースメントは16ビットの符号付整数として扱われるため，−32768〜+32767の範囲が指定できる．記述形式は d(An) である（n=0〜7）．d は絶対値式でなければならない．

```
[例]  DISP   EQU      4500
             ADD .W   DISP+8 (A1), D2
                      (この場合の d の値は 4508 となる)
```

11.4.7 インデックス付アドレス レジスタ間接形式

指定したアドレス レジスタとインデックス レジスタ，およびディスプレースメントを加えた値が指すメモリの内容が実行の対象となる．ディスプレースメント d は8ビットの符号付整数であり，−128〜127の範囲が指定できる．記述形式は d (An, Ri) または d (An, Ri .W) または d (An, Ri .L) である (n=0〜7)．Ri はインデックス レジスタであり，アドレス レジスタ Ai および データ レジスタ Di の中から任意の1個をこれに当てることができる (i=0〜7) .W または .L を指定することにより，インデックスレジスタの下位16ビットを使うか，32ビットすべてを使うかを選ぶことができる．何も指定しないときは .W が選ばれる .W を指定したときは16ビット目（最上位ビット）を符号拡張したものが使われる．d は絶対値式でなければならない．

```
[例]  DISP   EQU      14
             AND .B   D0, 60 (A1, A2 .L)
             OR .L    DISP−70 (A0, D1 .W), D5
                      (この場合の d の値は −56 となる)
```

11.4.8 絶対アドレス形式

オペランド フィールドの値が指すメモリの内容が実行の対象となる．記述形式は m である．ここで，m は絶対値式または相対値式である．この命令はアドレス空間の全領域を指定できる．もし 0～32767 までのアドレスが指定された場合には短絶対アドレス形式となるため，命令の長さを短くすることができる．

［例］　ADR 1：絶対値が EQU でセットされている．

```
ADD .W    2154, D1
ADD .W    ADR 1, D1
ADD .W    *+ADR 1, D1
```

11.4.9 ディスプレースメント付プログラムカウンタ相対形式

オペランド フィールドで指定したラベル，あるいは式の値が指すメモリの内容が実効アドレスとなる．記述形式は m であり，m には相対値式が入る．命令の中のディスプレースメントは 16 ビットの符号付整数であるから，この形式で指すことのできるメモリの範囲は，命令の先頭アドレス（＝＊）に対して －32766～32769 である．これはプログラムカウンタが＊＋2 を指しているためである．

［例］　ADR 1：EQU で与えられた絶対値
　　　 LBL 1：実行命令のラベル

```
DBNE    *+ADR 1
BRA     LBL 1
```

11.4.10 インデックス付プログラム カウンタ相対形式

プログラム カウンタと，インデックス レジスタと，ディスプレースメントを加えた値が指すメモリの内容が実効アドレスとなる．記述形式は d（Ri）または d（Ri .W）または d（Ri .L）である．インデックス レジスタ Ri の指定のしかたはインディクス付アドレス レジスタ間接形式の場合と同じである．d には相対値式を書くが，式の値は命令の先頭アドレスに対して －126～＋129 の範囲に入っていなくてはならない．

［例］　LBL：ラベル

```
SUB .L    LBL+10 (A0), D2 .W
```

11.4.11 イミディエイト データ形式

オペランドがそのまま実行の対象となる．記述形式は # m である． m は絶対値式，文字定数，あるいはアドレスの確定しているプログラムの中の相対値式である．指定できる値の大きさは，オペランド フィールドで指定するデータ サイズにより決る．

［例］　ADR 1 : EQU で定義された絶対値

```
MOVE .W   #100, D1       10進数
MOVE .L   #'AXZ3', D1    文字定数
MOVE .W   #ADR1, D1      絶対値式
```

11.4.12 CCR/SR 形 式

ステータス レジスタ (SR) またはコンディション コード レジスタ (CCR : SR の下8ビット) が実効アドレスとなる．記述形式は SR あるいは CCR である．この形式は ANDI, ORI, EORI の第2オペランド，および MOVE で使うことができる．

［例］

```
ANDI .B   #$1D, CCR
MOVE      SR, D0
```

11.5 アセンブラ制御命令

ここではいろいろなアセンブラ制御命令のうちで基本的なものについて説明する．

（1） ORG　〈絶対アドレス〉(Absolute Section Origin)

オペランドフィールドに書かれた値を ORG に続くステートメントの絶対アドレスとする．この命令はプログラムの途中に複数回現れてもよい．この命令にラベルをつけることはできない．

［例］　ORG　$1000

（2）〈記号〉　EQU　〈式〉(Equate Symbol Value)

ラベル フィールドの記号に式の値を与える．式は絶対値式がふつうであるが，すでに現れた記号あるいはロケーション カウンタ (*) を用いた相対値式も記述できる．ラベルを省略することはできない．

［例］

```
EAST   EQU   3
WEST   EQU   EAST
HERE   EQU   *
```

（3） DC 〈式〉,〈式〉, …(Define Constant)

式を計算した値をメモリに定数データとして格納する命令であり，カンマによって複数のデータ定義が可能である．DCに続いてデータサイズを .B, .W, .L で定義でき，省略時には .W が仮定される．この命令にはラベルをつけてもよい．

[例] VAL1 DC .W 31, $A, 'XYZ'

DC .B 'ABC'

（4） DS 〈絶対値式〉(Define Storage)

絶対値式で与えられた大きさのエリアを，データ サイズを単位としてメモリに確保する．データ サイズは .B, .W, .L で指定でき，省略時には .W が仮定される．命令にラベルをつけることが可能である．

[例] AREA DS .L 8

これまでに述べたアセンブラ制御命令および実行命令を用いたプログラム例を図11.4に示す．この例は先頭が TBL1 で示されるテーブル1を，先頭が TBL2 で示されるテーブル2に加算するものである．またテーブル1のデータが0になったらそこで加算を終了する．MCOUNT はテーブル1の値が0にならない場合に終了すべき回数を設定している．ここではアセンブラ制御命令として EQU, ORG, DS, DC を用いており，このほかにプログラムの開始を示す NAM, 終了を示す END を使っている．

```
                            00001           NAM     SAMPLE1
                            00002   *********************************************
                            00003   *  SAMPLE PROGRAM   ( ADD TABLE )           *
                            00004   *       TABLE1 + TABLE2 --> TABLE2          *
                            00005   *********************************************
        0000 0014           00006   MCOUNT  EQU     20          MAX TABLE LENGTH
        0000 2000           00007   TBL2    EQU     $2000       TABLE2 ADDRESS
        0000 000A           00008   MONITOR EQU     10          MONITOR TRAP NO.
                            00009   *
        0000 1000           00010+++        ORG     $1000       PROGRAM ADDRESS
                            00011   *
001000  227C 0000 102A      00012           MOVE.L  #TBL1,A1    LOAD TABLE1 ADDRESS INTO A1
001006  247C 0000 2000      00013           MOVE.L  #TBL2,A2    LOAD TABLE2 ADDRESS INTO A2
00100C  323C 0014           00014           MOVE.W  #MCOUNT,D1  LOAD MAX COUNT INTO D1
001010  4242                00015           CLR.W   D2          INITIALIZE D2(COUNTER)
001012  3619                00016   START   MOVE.W  (A1)+,D3    LOAD TABLE1 INTO D3
001014  6700 000A           00017           BEQ     FINISH      IF DATA = 0 THEN END
001018  5242                00018           ADDQ.W  #1,D2       COUNTER INCREMENT
00101A  D75A                00019           ADD.W   D3,(A2)+    ADD AND STORE TABLE2
00101C  51C9 FFF4           00020           DBRA    D1,START    MAX COUNT CHECK
                            00021   *
001020  33C2 0000 1028      00022   FINISH  MOVE.W  D2,COUNT    STORE COUNT VALUE
001026  4E4A                00023           TRAP    #MONITOR    GO TO MONITOR ROUTINE
                            00024   *
001028  0000 0002           00025   COUNT   DS.W    1
                            00026   *
00102A  0087 0056 01C8      00027   TBL1    DC.W    135,86,456,8,2,127,87,54,0
001030  0008 0002 007F
001036  0057 0036 0000
                            00028           END
```

図 11.4 ソース プログラムのアセンブル例

11.6 リロケータブルなプログラミング

これまで述べてきたプログラムは，ソースプログラムを記述するときにすでにアドレスが確定しているものと考えていた．しかし大きなシステムでは，プログラムをいくつかの単位に分割して作っておき，これらを適当に組み合せて所望のプログラムを作る場合が多い．この利点は，一般に機能単位でプログラムを分割するので，複数のプログラマが同時に作業できること，デバッグや修正が容易であること，別のプログラムへの変更も単位プログラムの組合せを変えるだけで対応できること，などである．この場合ソース プログラムでは絶対アドレスを与えず，どこにでも配置できるようにしておくことが望ましい．このようにすることをリロケータブルなプログラミングとよぶ．

リロケータブルなプログラムの分割の単位はセクションである．セクションには番号がつけられており，各セクションはアセンブラにより仮のアドレスがついたオブジェクトモジュールになる．そしてリンケージ エディタで，先頭のアドレスと，セクションの番号を結合したい順に指定すると，絶対アドレスの確定したロード モジュールが作成される．この様子を図 11.5 に示す．(a) はソースプログラムおよびこれをアセンブルしたオブジェクト モジュールにおけるセクションの並びを示している．セクション 3 のように 1 つのセクションがさらに分割されていてもよい．リンケージ エディタで，セクション 1 と 2 を 2, 1 の順に ＄1000 番地から割り当て，セクション 3, 4, 5, は 5, 3, 4 の順に ＄4000 番地から割り当てるように指定する．するとロード モジュールは図の (b) のように，アドレス付

＃3 (A)
＃5
＃3 (B)
＃1
＃2
＃3 (C)
＃4

(a) リロケータブル オブジェクト モジュール

＄1 000 →

＃2
＃1

＄4 000 →

＃5
＃3 (A) (B) (C)
＃4

(b) ロード モジュール
(例：＄1000 からセクション 2 と 1 を割当て
＄4000 からセクション 5 と 3 と 4 を割当て)

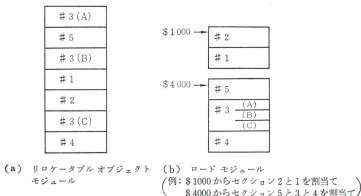



図 11.5 メモリの割当て例，注) ＃n はセクション番号

けされて作成される．

リロケータブルなプログラムにするには新たなアセンブラ制御命令が必要である[†]．リロケータブルなセクションであることを宣言するには，ORG 命令の代りに SCT あるいは SECTION 命令を用いる．またリロケータブルなセクションでは，ベースレジスタを登録しておきベースレジスタ間接の方法で，アドレス指定をする．この場合まず USING 命令でどのアドレスレジスタをベースレジスタとするかをアセンブラに対し指定する．するとラベルを参照するアドレス形式は自動的にディスプレースメント付アドレス レジスタ間接形式，あるいはインデックス付アドレス レジスタ間接形式に変換される．なお USING 命令は，アセンブラ制御命令でありレジスタに実際の値を入れる機能はない．ベースレジスタに値を設定する命令はユーザが実行命令で記述しておかねばならない．リロケータブルにするための命令はこのほかにもいくつかあるが，ここではふれない．

† 小規模なアセンブラではサポートしていないものもある．

12 データ転送命令

データ転送命令は，メモリや68000の内部レジスタ間でデータを転送するのに用いられる．68000では，特に多くの種類のデータ転送命令を有していることが命令セットの特徴ともなっている．さらにバイト，ワード，ロング ワードなどのオペランド サイズと14種のアドレス形式を組み合せることにより，高速データ転送，大量データ転送など効率の良いデータ転送を実現できる．5章では各転送命令がとり得るオペランド サイズとオペレーションの概要を示した．この章では，転送命令のうち，LINK，UNLK命令以外の命令とその主要な応用例について説明する．LINK，UNLK命令は15章で述べる．

12.1 データ転送命令

a．MOVE（Move Data）命令†　MOVE命令はデータ転送命令のなかで最も基本的な命令である．この命令は，バイト（B），ワード（W），ロング ワード（L）のデータを図12.1のようにメモリ，レジスタ間で転送する．このようにメモリ，レジスタの任意の組合せで転送ができること，付録の命令一覧表で示すように，ソースおよびデスティネーションのアドレス形式を独立に，かつ任意に指定できることが大きな特徴となっている．

MOVE命令ではデータ転送時にデータの正，負，0を判定して，その結果をコンディション コード レジスタのN，Zの各フラグに反映させる．一方，V，Cの各フラグは，リセットされる．X（Extend）フラグは影響を受けない．

データ レジスタからメモリへのデータ転送のための命令の使用例を以下に示す．

---

† MOVE　〈ea 1〉，〈ea 2〉　〈ea 1〉⟶〈ea 2〉

		デスティネーション				
		メモリ	データレジスタ	アドレスレジスタ	コンディションコードレジスタ	ステータスレジスタ
ソース	メモリ	B W L	B W L	W L (MOVEA)	W (MOVE to CCR)	W (MOVE to SR)
	データレジスタ	B W L	B W L	W L (MOVEA)	W (MOVE to CCR)	W (MOVE to SR)
	アドレスレジスタ	B W L	B W L	W* L (MOVEA)		
	イミディエイトデータ	B W L	B W L	W L (MOVEA)	W (MOVE to CCR)	W (MOVE to SR)
	コンディションコードレジスタ					
	ステータスレジスタ	W (MOVE from SR)	W (MOVE from SR)			

B: バイト，W: ワード，L: ロングワード

図 12.1　データ転送命令のソースとデスティネーション

```
MOVE .B   D0, D1        D0 の内容の最下位バイトを D1 へ転送する
MOVE .L   D2, ADDR      D2 の内容を ADDR 番地へ転送する
MOVE .W   D3, (A1)+     D3 の内容を A1 で指定されたメモリの番地へ
                        転送し，その後 A1 を +2 する
```

ここで ADDR と記述されたラベルには，あらかじめメモリ番地を与えておき，アドレス レジスタ A1 にも，あらかじめメモリ番地がセットされているとする．

b. MOVEA (Move Address) 命令†　この命令はアドレスデータの転送に用いられるため，転送のデスティネーションはつねにアドレス レジスタとなる．転送データのサイズは，ワード，ロング ワードのみである．各フラグは影響を受けない．

MOVEA 命令の機械語は，MOVE 命令においてデスティネーション オペランドをアドレス レジスタに指定したときのものと同じである．このため MOVE 命令でデスティネー

† MOVEA 〈ea〉, An　〈ea〉⟶An

ションオペランドにアドレスレジスタを指定すると，アセンブラにおいて MOVEA 命令に自動的に置き換る．なおデータサイズがワードの場合はデータの符号拡張が行われる．

本命令の例として，サブルーチンジャンプ時のレジスタの退避処理を次に示す．

```
        :
        JSR       SUBR         サブルーチン SUBR へ分岐する
        :
SUBR    MOVEA.L   A0, SAVE     A0 の内容を SAVE 番地へ退避する
        :
        MOVEA.L   SAVE, A0     SAVE 番地より A0 の値を回復する
        RTS                    サブルーチンからの復帰
```

MOVE 命令と MOVEA 命令におけるデータ転送の様子をまとめて図 12.2 に示す．（1）はアドレスレジスタで指定されたメモリ上の 32 ビットのデータをデータレジスタに転送する場合である．ADDR i 番地のデータはデータレジスタの上位ワードへ，ADDR i+2 番地のデータは下位ワードに移される．（2）はメモリ間のデータ転送の場合でアドレスレジスタを 2 個使って任意のメモリ間のデータ転送が行える．（3）はアドレスレジスタ

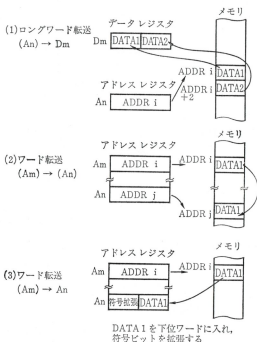



図 12.2　データ転送命令の例

へのデータ転送の場合（MOVEA 命令）で，32ビット データに符号が拡張されていることに注意されたい．

c. MOVEM（Move Multiple Registers）命令†1　MOVEM 命令は，図 12.3 に示されるように，命令語の第2ワードで指定されるレジスタ リストと第1ワードの実効アドレスを用いて，レジスタとメモリ間のデータ転送を行う．図 12.3 のレジスタ リスト上の各ビットの“1”指定により選択されたデータ レジスタおよびアドレス レジスタと，実効アドレスを先頭アドレスとしたメモリとの間でデータ転送が行われる．レジスタリスト上の各ビットで選択できるレジスタは，図 12.3 で示すようにメモリのアドレス更新方向によって異なるので注意が必要である．これについては後述する．この命令では，オペランド サイズをワード，およびロング ワードとすることができる．特にオペランドサイズとしてワ

命令コーディング例	命令コーディング例

MOVEM (A6),A4/A5/D5-D7
または
MOVEM (A6)+,A4/A5/D5-D7
	MOVEM −(A6),A4/A5/D5-D7

命令語第1ワード：MSB ----------- LSB

命令語第2ワード（レジスタ リスト）

A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0

D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A6	A7
0	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0

← レジスタ指定順序（1：選択　0：非選択）

メモリ

実行前の実効アドレス → D5へ転送 / D6へ転送 / D7へ転送 / A4へ転送 / A5へ転送 → 実行後の実効アドレス（アドレス更新 ↓）

実行後の実効アドレス → D5へ転送 / D6へ転送 / D7へ転送 / A4へ転送 / A5へ転送 ← 実行前の実効アドレス（アドレス更新 ↑）

図 12.3 MOVEM 命令とレジスタ リスト

---

†1 MOVEM 〈レジスタ リスト〉，〈ea〉　レジスタ リストで指定されたレジスタ⟶〈ea〉
　MOVEM 〈ea〉，〈レジスタ リスト〉　〈ea〉⟶レジスタ リストで指定されたレジスタ

†2（次ページ）一般の MOVE 命令では，オペランド長がワードのときにはデータレジスタは符号拡張されないので注意されたい．

ードを指定しデスティネーションにデータ レジスタを指定した場合，データ レジスタには 32 ビットに符号拡張されたデータが転送される[†2](前ページ)．

MOVEM 命令は，アドレス形式により，その動作形態が若干異なるので，それらについて以下に説明する．

（i） アドレス レジスタ間接，ディスプレースメント付/インデックス付アドレス レジスタ間接および長/短絶対アドレス形式　図 12.3 に示すように，レジスタ データの転送は D0, D1, …, D7, A0, A1, …, A7 の順序で，メモリ データは命令で指定された先頭実効アドレスからアドレスが増加する順序で，データ転送が行われる．転送方向は，「レジスタからメモリへ」，および「メモリからレジスタへ」の双方向が可能である．

アドレス レジスタ間接，ディスプレースメント付アドレス レジスタ間接の各アドレス形式を用いてメモリからレジスタへの転送を行う場合，および実効アドレスを指定しているアドレス レジスタ自身にメモリからデータを転送した場合について説明する．68000 では MOVEM 命令実行開始時に，上記アドレス レジスタの内容を内部の一時記憶用の特別なレジスタに移しておき，この一時記憶レジスタを用いて実効アドレス計算を行っている．したがって実効アドレスを指定しているアドレス レジスタへのデータ転送も矛盾なく行われる．

（ii） ディスプレースメント付/インデックス付プログラムカウンタ相対アドレス形式
レジスタおよびメモリのデータ転送の順序は（i）と同じである．しかしデータ転送の方向は，「メモリからレジスタへ」の転送のみ可能である．実効アドレスを記憶しているアドレス レジスタ（本命令ではデータ レジスタも含まれる）に，メモリよりのデータを書き込む場合は，（i）と同じ理由で正常にデータ転送が行われる．

（iii） プリデクリメント アドレスレジスタ間接アドレス形式　このアドレス形式では，前記の場合とは逆に，レジスタデータの転送は A7, A6, …, A0, D7, D6, …, D0 の順序である．メモリ データの転送は指定された実効アドレスからワードの場合は 2，ロング ワードの場合は 4 を減じたアドレスを先頭としてアドレスが減少する順序で行われる．転送方向は，「レジスタからメモリへ」のみが可能である．命令実行完了時の実効アドレス用のアドレスレジスタには，最後にデータを書き込んだメモリ番地が残っている．

（iv） ポストインクリメント アドレス レジスタ間接アドレス形式　データ転送が行われていくにつれて，実効アドレスが増加していくのは（i）と同様であるが，データ転送の方向は，「メモリからレジスタへ」のみが可能である．

このアドレス形式を用いて，実効アドレスを記憶しているアドレス レジスタへメモリか

らデータを転送すると，命令実行終了時には最後に読み出したメモリ アドレスに2を加えたアドレス（ロング ワードのときは4を加えたアドレス）が書き込まれる．

MOVEM 命令の使用例を以下に示す．

［例1］ メモリの$1000番地から，8バイトのデータを，図12.4のようにD0, D3, A0, A1の各レジスタへ1ワードずつ転送する．ただしA0は実効アドレスを定めるアドレス レジスタとして用いる．MOVEM命令実行後に各レジスタに転送されたデータは符号拡張されたものとなっている．

```
MOVEA .L   #$1000,A0
MOVEM .W   (A0),D0/D3
           /A0-A1
ORG        $1000
DC .L      $80819091
DC .L      $A0A1B0B1
```

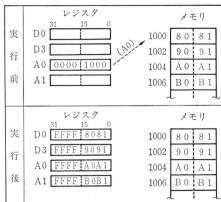



図 12.4 MOVE 命令実行例

［例2］ ポスト インクリメント付レジスタ間接アドレス形式を用いると，アドレス レジスタA0には，命令実行の途中で一旦，$FFFFA0A1の値が記憶されるが，命令終了時には，最終実効アドレス$00001008が記憶される．

```
MOVEA .L   #$1000,A0
MOVEM .W   (A0)+,D0/D3/A0-A1
ORG        $1000
DC .L      $80819091
DC .L      $A0A1B0B1
```

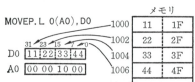
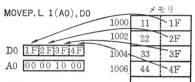
データ転送後のレジスタには以下の値がセットされる．

(D0)=$FFFF8081
(D3)=$FFFF9091
(A0)=$06001008
(A1)=$FFFFB0B1

d. MOVEP (Move Peripheral Data) 命令† MOVEP命令は，データ レジスタとメモリの間で，バイト単位でデータ転送を行う．指定できるオペランド サイズはワードおよ

† MOVEP Dx, d(Ay) データ レジスタ Dx→d(Ay)
MOVEP d(Ay), Dx d(Ay)の内容→データ レジスタ Dx

びロング ワードであるので，バイト データを数回に分けて転送する．すなわち，ワードの場合は2回，ロング ワードの場合は4回である．

MOVEP 命令におけるメモリ実効アドレスはディスプレースメント付アドレス レジスタ間接形式のみ指定できる．データ転送は，データ レジスタに対しては上位バイトから順に，メモリに対しては上記アドレス形式で指定される先頭アドレスから2番地ずつ増加するような順序で行われる．本命令ではバイト データの転送を行うために，オペランド サイズがワードおよびロング ワードであるにもかかわらず，メモリのデータ転送開始アドレスは，奇数および偶数のいずれでもよい．本命令は6800の周辺LSIのような8ビット データ バスに接続されるデバイスとのデータ転送に有効である．

(a) 偶数アドレスの場合

(b) 奇数アドレスの場合

図 12.5 MOVEP 命令実行例

図12.5に示すように，メモリのアドレス指定が，偶数と奇数の場合のプログラム例を以下に示す．

(i) 偶数アドレスの場合(図12.5(a))　この例では実効アドレスの先頭は$1000番地である．メモリの偶数アドレスの内容(各ワードの上位バイト)が4回に分けて，データ レジスタ D0 の32ビットに転送される．なおオペランド サイズがワードの場合は，$1000と$1002番地の2バイトがD0の下位に格納される．

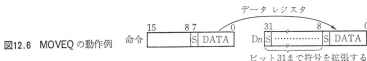
```
MOVEA .L    #$1000,A0
MOVEP .L    0(A0),D0
```

(ii) 奇数アドレスの場合 (図12.5 (b))　この例では実効アドレスの先頭は$1001番地である．メモリの奇数番地の内容(各ワーバの下位バイト)が4回に分けて，データ レジスタ D0 の32ビットに転送される．なおオペランド サイズがワードの場合は，$1001と$1003番地の2バイトがD0の下位ワードに格納される．

```
MOVEA .L    #$1000,A0
MOVEP .L    1(A0),D0
```

e. MOVEQ (Move Quick) 命令†　MOVEQ 命令は，図12.6に示すようにオペレーションワードのなかに含まれる8ビットのイミディエイト データを データ レジスタへ転

† MOVEQ　#〈データ〉, Dn　　#〈データ〉⟶Dn

図12.6 MOVEQの動作例

送するものである．このとき32ビットに符号拡張してデータ レジスタへ転送する．イミディエイト データは符号付の8ビット データとして表現されるため，10進数では −128 ～+127 の範囲の値をとり得る．

MOVE および MOVEQ のイミディエイト アドレス形式の実行の比較を以下に示す．なお ＊はこの命令によって値が変化しない部分を示す．

```
MOVE .L   #$80,D0      (D0)=$00000080; 符号拡張なし
MOVE .B   #$80,D1      (D1)=$******80; 符号拡張なし
MOVEQ     #$80,D2      (D2)=$FFFFFF80; 符号拡張あり
MOVE .L   #-128,D3     (D3)=$FFFFFF80; 符号拡張なし
MOVE .B   #-128,D4     (D4)=$******80; 符号拡張なし
MOVEQ     #-128,D5     (D5)=$FFFFFF80; 符号拡張あり
```

アセンブラでは，MOVE .L 命令でイミディエイト値が −128～127 の範囲であると，MOVE .L 命令の代りに MOVEQ 命令のオペレーション コードを割り当てる．すなわち今述べた例では4番目と最後の命令の転送データは等しい．MOVE .L 命令と MOVEQ 命令を比較すると，バイト数と実行サイクル数がそれぞれ，6バイトと12クロック サイクル，2バイトと6クロック サイクルであるため，MOVEQ 命令の方がバイト数，サイクル数ともに有利であることがわかる．なお MOVEQ 命令は MOVE 命令と同様，コンディション コード レジスタに影響を与える．

f． LEA (Load Effective Address) **命令**† LEA は，ソース オペランドの実効アドレス値を計算してその結果をデスティネーション オペランドで指定されるアドレス レジスタにセットするものである．デスティネーションのアドレス レジスタの32ビットはすべて影響を受ける．次に命令の使用例 (1)，(2)，(3) を示す．

（1） 本命令の先頭アドレスから16バイト増加したアドレス値を A0 にセットする．なお，この命令自身が4バイト長である．

```
LEA    *+$10,A0
```

（2） A1 の内容に +2 して，A1 に再格納する．

```
LEA    2(A1),A1
```

† LEA 〈ea〉, An　ea で表されるアドレス ⟶ An

（3） 16進の1000番地をアドレス レジスタ A3にセットする．

LEA　　$1000, A3

g. PEA (Push Effective Address) 命令[†1]　PEA命令は，デスティネーションオペランドの実効アドレスを計算して求め，その結果をシステム スタックに格納する．転送されるアドレス データは必ずロング ワードである．例を次に示す．

（1） アドレス レジスタ A5の内容をスタックに退避する．

PEA　　(A5)

（2） アドレス レジスタ A6の内容を +2して，スタックに退避する．

PEA　　2(A6)

（3） 16進の1000番地をアドレス データとしてスタックに退避する．

PEA　　$1000

h. EXG (Exchange Registers) 命令[†2]　EXG命令は，次のように2つのレジスタ間で，その内容を相互に交換する．

（1） データ レジスタ　―データ レジスタ（図12.7参照）

（2） アドレス レジスタ―アドレス レジスタ

（3） データ レジスタ　―アドレス レジスタ

データ レジスタ
Dn　DATA 1
Dm　DATA 2

図 12.7 EXG命令（レジスタ間のデータ交換を行う）

本命令のオペランド サイズは ロング ワードのみである．

2つのレジスタ間で内容の交換を行う場合，本命令を用いると，命令バイト数，処理時間，一時記憶領域が不要などの面から有利となる．

i. SWAP (Swap Register Halves) 命令[†3]　SWAP命令は図12.8に示すように，指定されたデータ レジスタの上位ワード（16ビット）と下位ワード（16ビット）の内容を入れ替える．コンディション コード レジスタはMOVE命令と同様に演算結果によって影響を受ける．割算を例として SWAP命令の使い方を次に示す．

図 12.8 SWAP命令（同一レジスタ内の上位ワードと下位ワードのデータ交換を行う）

DIVS命令を実行すると，データ レジスタの下位ワ

†1 PEA ⟨ea⟩　　eaで表されるアドレス⟶システム スタック
†2 EXG Rx, Ry　　RxとRyの内容を交換する．
†3 SWAP Dn　　Dnの上位ワードと下位ワードを入れ替える．

ードに商，上位ワードに余りが格納される．これを分離して別のメモリ番地（商のアドレス＝QUONT，余りのアドレス＝RMNDR）へ移す．

```
MOVEQ     #-13,D2       (D2)=-13
DIVS      #5,D2         -13÷5
MOVE.W    D2, QUONT     商を QUONT へ格納する
SWAP      D2            D2 の上位，下位ワードの入替えを行う
MOVE.W    D2, RMNDR     余りを RMNDR へ格納する
```

12.2 データ転送命令の応用例

12.2.1 単純なデータ転送

ここでは単純なデータ転送の例を示す．オペランド サイズとしてバイト，ワード，ロング ワードのものを使用した．また，アドレス形式としては，データ レジスタ直接形式，アドレス レジスタ直接形式，アドレス レジスタ間接形式，ディスプレースメント付/インデックス付アドレス レジスタ間接形式，イミディエイト データ形式を適宜組み合せた．データレジスタ，アドレスレジスタ，メモリの間で，種々のサイズのデータを，いろいろなアドレス形式を使用して転送している．

```
          ORG       $1000             開始アドレス=$1000
          MOVE.B    DATA1,D0          $04→D0
          MOVE.W    DATA2,D1          $1020→D1
          MOVE.L    DATA3,D2          $30405060→D2
          MOVEA.W   #$10FF,A0         $000010FF→A0
          MOVE.B    D0,(A0)           $04→$10FF 番地
          MOVE.W    D1,1(A0)          $10→$1100 番地
                                      $20→$1101 番地
          MOVE.L    D2,-1(A0,D0)      $30→$1102 番地
                                      $40→$1103 番地
                                      $50→$1104 番地
                                      $60→$1105 番地
DATA1     DC.B      $4                DATA1 の定数定義
DATA2     DC.W      $1020             DATA2 の定数定義
DATA3     DC.L      $30405060         DATA3 の定数定義
```

12.2.2 ブロック転送

まとまった量のデータ転送をブロック転送とよぶことが多い．68000 では 16M バイトまでのブロック転送を数命令で実行できる．しかし実際には，大量のデータを転送する場

合には，より高速転送のできるダイレクト メモリ アクセス法（DMA）を利用する場合が多い．したがって命令によるブロック転送は，比較的少量のデータ転送に適している．以下にブロック転送プログラムの例を示す．

```
        MOVEA .L   SPOINT , A 0      ソースのポインタを設定する
        MOVEA .L   DPOINT , A 1      デスティネーションポインタを設定
                                     する
        MOVE .L    COUNT , D 0       転送語数を設定する
LOOP    MOVE .W    (A 0)+ , (A 1)+   データ転送を行う
        DBRA       D 0 , LOOP        D0 を 1 減ずる．結果が −1 でない
                                     と LOOP へ分岐する
SPOINT  DC .L      $10000            ソース ポインタのスタート アドレ
                                     ス
DPOINT  DC .L      $20000            デスティネーション ポインタのス
                                     タート アドレス
COUNT   DC .L      1000              転送語数
```

この例では，＄10000 番地から始まる 1001 語のデータ ブロックを，＄20000 番地から始まる領域へ転送する．MOVE 命令の両オペランドにはポスト インクリメント付アドレスレジスタ間接形式を用いている．このプログラムは，22 バイトのプログラム サイズである．また，1 ワードの転送時間は 23 クロック サイクルである．68000 のクロック周波数を 8 MHz とした場合，このワード転送に要する時間は 2.875 μs となる．なお，転送語数が偶数ワードの場合には前の例の LOOP とラベル付けされた MOVE 命令のオペランド サイズをロング ワードにすると，ブロック転送をより速く実行できる．

12.2.3 プログラム スタック（Program STACK）

68000 には，2 章で述べたように，システム スタックとしてスーパバイザ システム スタックとユーザ システム スタックがある．これらのスタック ポインタとして，アドレス レジスタ A7 を割り当てている．本節ではこのようなシステム スタックではなく，プログラムの中で必要に応じて作ることができるプログラム スタックの構成について説明する．プログラムスタックを実現するためには，アドレス レジスタ A0 から A6 の 7 本のアドレス レジスタのうちの 1 つを使用してポスト インクリメント，あるいはプリデクリメントアドレス レジスタ間接のアドレス形式を伴うデータ転送命令を用いる．メモリ番地が減少する方向にプログラム スタックを構成する場合，プッシュ（データの格納）としてプリデクリメント アドレス レジスタ間接形式の MOVE 命令を，ポップ（データの再読出し）にはポスト インクリメント アドレス レジスタ間接形式の MOVE 命令を用いる．逆にメ

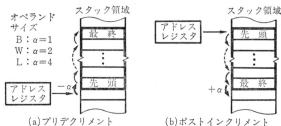



図 12.9　スタック操作上の注意

モリ番地が増加する方向にプログラム スタックを構成する場合には MOVE 命令のアドレス形式としてプッシュにはポスト インクリメント間接を，ポップにはプリデクリメント間接を用いる．

この操作を図12.9を用いて，以下に説明する．

（1）　プリデクリメント アドレス レジスタ間接形式の MOVE 命令において，68000 はアドレス レジスタを α の値だけデクリメントした後で，そのレジスタをスタックのポインタとして用いる．したがってアドレス レジスタにはスタックの先頭番地$+\alpha$ の値を初期設定しておく必要がある．ここで α はオペランド サイズがはバイトのとき1，ワードのとき2，ロング ワードのとき4である．

（2）　ポスト インクリメント アドレス レジスタ間接形式の場合，68000はアドレス レジスタをプログラム スタックのポインタとして用いた後，インクリメントする．したがって命令実行後，レジスタの値には，α（α の値は (1) と同様）が加算される．

（3）　バイト データとワード データ（ロング ワード データも含む）とをプログラム スタック内に混在させて記憶する場合には，後にプログラムスタックからそのデータを再び読み出すときに配列誤りを起さないように，プログラム作成上の注意が必要である．すなわち，オペランド サイズがワードやロング ワードの場合には，実効アドレスが常に偶数になるようにしなければアドレス エラーが生ずる．したがってバイト データを扱う場合には，ダミーのバイト データを本来のバイト データに付加し，ワード データのようにして扱うような配慮も必要となる．

次にスタック形記憶の例として以下に数値計算の一方法を示す．

$$X=A+B\times(C+D)$$

これを逆ポーランド記法を用いて書き換えると次のようになる．

$$X=ABCD+\times+$$

メモリ番地が小さくなる方向に構成されたプログラム スタックを用いて，上式を実行するプログラムを作成すると次のようになる．

```
MOVEA .L   #STKP+2,A0      スタック ポインタ A0を初期設定する
MOVE .W    INPUT,-(A0)     Aをプログラム スタックに格納する
MOVE .W    INPUT,-(A0)     Bをプログラム スタックに格納する
MOVE .W    INPUT,-(A0)     Cをプログラム スタックに格納する
MOVE .W    INPUT,D0        DをD0に格納する
ADD .W     (A0)+,D0        C+D
MULS .W    (A0)+,D0        B(C+D)
ADD .W     (A0)+,D0        A+B(C+D)
```

この例では，外部入力装置（INPUT 番地に割り付けられている）から16ビットのデータを順次入力する．68000の入力動作に同期して外部入力機器は680000への出力動作をしていると仮定している．このプログラムでは数値の入力順にプログラム スタックでこれを記憶していくため，データ レジスタを用いないですむことや，アドレス形式にA0修飾を用いているので命令バイト数が少なくなるなどの特徴をもつ．68000はスタックに対して種々演算を行えるため，いわゆるスタック マシンと同様の機能が実現できる．

12.2.4 キュー（QUEUE）

キューはスタックと同様に，アドレスを示すポインタを更新しながらデータを格納したり読み出したりするものである．しかしスタックが，あとから格納したデータから順に読み出す（Last-in-First-out=LIFO）のに対してキューではさきに格納したデータから順に読み出す（First-in-First-out=FIFO）ことが特徴となっている．このようにキューでは格納した順序と同じ順序でデータを読み出すため，データ格納用ポインタとデータ読出し用ポインタの対が必要である．このように，プログラムでキューを構成するために，A0からA6までのアドレス レジスタのうち2本を使用する．メモリ番地が増加する方向にキューを構成する場合には，データ格納とデータ読出しを行うMOVE命令のアドレス形式として，共にポスト インクリメント アドレス レジスタ間接形式を用いる．逆にメモリ番地が減少する方向にキューを構成する場合には，共にプリデクリメント アドレス レジスタ間接のアドレス形式を用いる．

スタックはLIFOであるのでデータ格納とデータ読出しの頻度が同じ程度ならスタック領域に大きな増減はない．しかしキューはFIFOのため，データ格納が続くときキュー領域はそれに応じて増加し続ける．これを防ぐためには，次のようなキューの操作をプログ

ラムで工夫する必要がある．プログラムでキューの容量を指定して，データ格納用ポインタの内容がこの指定値を超えたら，データ格納用ポインタを初期値（キューの格納開始番地）に再設定する．同様に，データ読出しポインタの内容がキュー容量指定値を超えたら初期設定を行う．このようなデータ格納用ポインタとデータ読出し用ポインタを初期値に再設定しながら用いるキューを循環バッファとよぶ．循環バッファを構成する場合，データ格納用ポインタの内容とデータ読出しポインタの内容の差を常に監視して，いわゆるオーバランが生じないようなプログラム上の注意が必要である．

13　算術論理演算命令

この章では，各算術論理演算命令についてその機能を説明する．各命令のソース オペランドおよびデスティネーション オペランドに許されるアドレス形式については，付録の表を参照されたい．また各命令のコンディション コードの変化については6章を参照されたい．さらにオペランド サイズに関しては通常の命令ではバイト，ワード，ロング ワードの3種類が使用可能である．特に制限がある場合のみ説明を加える．

13.1 算術演算命令

算術演算命令には加算，減算，乗算，除算のほか補数，比較，テスト，エクステンド，クリア命令がある．以下各命令毎に説明する．

13.1.1 加算命令

加算命令には ADD，ADDA，ADDI，ADDQ，ADDX 命令の5種の命令がある．このうち ADDX は多倍精度加算命令であり，13.1.4項で説明する．

a． ADD (Add Binary) **命令**† この命令はデータの加算に使用する命令であり，デスティネーション オペランドの内容にソース オペランドの内容を加算し，結果をデスティネーションに格納する．

デスティネーション オペランドにアドレス レジスタを指定すると，命令の機能がコンディション コードの変化も含めて ADDA 命令に一致する．

---

† ADD 〈ea〉, Dn　　Dn+〈ea〉⟶Dn
　ADD Dn, 〈ea〉　　〈ea〉+Dn⟶〈ea〉

b. ADDA (Add Address) 命令[†1]　この命令はアドレス データの加算に使用する命令である．デスティネーション オペランドで指定したアドレス レジスタ An にソース オペランドの内容を加算し，結果をアドレス レジスタ An に格納する．オペランド サイズはワードおよびロング ワードに制限されておりバイトは指定できない．ワード長の演算が指定された場合にはソース オペランドの符号ビット（ビット 16）を拡張してロングワードデータとして加算を行う．

ADDA 命令はアドレス データを加算するための命令なので，コンディション コードは変化しない．

c. ADDI (Add Immediate) 命令[†2]　この命令はデスティネーション オペランドの内容にイミディエイト データを加算し，結果をデスティネーションに格納する命令である．オブジェクト コードではイミディエイト データが命令ワードの次に付加される．デスティネーション オペランドにアドレス レジスタを指定することはできない．アドレス レジスタにイミディエイト データを加算するには ADDA 命令を用いる．

d. ADDQ (Add Quick) 命令[†3]　この命令は ADDI 命令と同様にイミディエイト データの加算をする命令であり，データに対する加算とアドレス データに対する加算のどちらにも使うことができる．イミディエイト データの範囲は 1 から 8 までに限定されているが，この範囲のデータなら ADDI 命令より ADDQ 命令のほうが命令バイト数が少なく，命令実行時間が短い．たとえばデータ レジスタへ加算する場合の命令バイト数と命令クロック サイクル数を比較すると次のようになる．

	命令長	実行時間
ADDQ .W　#2,D0	2バイト	4クロック サイクル
ADDI .W　#2,D0	4バイト	8クロック サイクル

クイック命令ではイミディエイト データが命令ワード中の 3 ビットで示されるために命令バイト数が少ない．またイミディエイト命令と異なりイミディエイト データを読み出すサイクルが不要なため実行時間が短い．

オペランド サイズはアドレス レジスタを使うときにのみワードまたはロング ワードに制限される．

---

†1　ADDA　<ea>, An　　An+<ea> ⟶ An
†2　ADDI　#<データ>, <ea>　　<ea>+#<データ> ⟶ <ea>
†3　ADDQ　#<データ>, <ea>　　<ea>+#<データ> ⟶ <ea>

13.1.2 減 算 命 令

減算命令には SUB, SUBA, SUBI, SUBQ, SUBX 命令の5種の命令がある．これらの命令のアドレス形式，コンディション コード，オペランド サイズは，それぞれ対応する加算命令の場合と一致する．

なお SUBX 命令は 13.1.4 項で説明する．

a． SUB (Subtract Binary) **命令**†1　この命令はデータの減算に使用する命令である．デスティネーション オペランドの内容からソース オペランドの内容を減算し，結果をデスティネーションに格納する．

デスティネーション オペランドにアドレス レジスタを指定すると命令の機能がコンディション コードの変化も含めて SUBA 命令に一致する．

SUB 命令を実行すると，演算結果によりコンディションコードが設定される．C フラグはボローが発生するとセットされ，それ以外のときはクリアされる．他のフラグの変化は ADD 命令と等しい．

b． SUBA (Subtract Address) **命令**†2　この命令はアドレス データに対する減算に使用する命令である．デスティネーション オペランドで指定したアドレス レジスタ An の内容からソース オペランドの内容を減算し，結果をアドレス レジスタ An に格納する．オペランド サイズはワードおよびロング ワードに制限されておりバイトは指定できない．ワード長の演算が指定された場合にはソース オペランドの符号ビット（ビット16）を拡張してロング ワード データとして減算を行う．

SUBA 命令ではコンディションコードは変化しない．

c． SUBI (Subtract Immediate) **命令**†3　この命令はデスティネーション オペランドの内容からイミディエイトデータを減算し，結果をデスティネーションに格納する命令である．オブジェクト コードではイミディエイト データが命令ワードの次に付加される．デスティネーション オペランドにアドレス レジスタを指定することはできない．アドレス レジスタからイミディエイト データを減算するには SUBA 命令を用いる．

d． SUBQ (Subtract Quick) **命令**†4　この命令は SUBI 命令と同様にイミディエイト

†1	SUB	〈ea〉, Dn	Dn－〈ea〉⟶Dn
	SUB	Dn, 〈ea〉	〈ea〉－Dn⟶〈ea〉
†2	SUBA	〈ea〉, An	An－〈ea〉⟶An
†3	SUBI	〈データ〉, 〈ea〉	〈ea〉－#〈データ〉⟶〈ea〉
†4	SUBQ	〈データ〉, 〈ea〉	〈ea〉－#〈データ〉⟶〈ea〉

データを減算する命令であり，データに対する減算とアドレス データに対する減算のどちらにも使うことができる．イミディエイト データの範囲は1から8までに限定されているが，この範囲のデータならSUBI命令よりSUBQ命令のほうが命令バイト数が少なく，命令実行時間が短い．アドレス レジスタを使うときオペランド サイズはワードまたはロング ワードに制限される．

13.1.3 補数命令

補数命令にはNEG, NEGX命令の2種の命令がある．NEGX命令は次の多倍精度演算の項で説明する．

NEG (Negate Binary) 命令†1 この命令はデスティネーション オペランドの内容をゼロから減算し，デスティネーションへ格納する命令である．

コンディション コードの設定はSUB命令と同様に行われる．

13.1.4 多倍精度演算命令

68000ではデータ レジスタが32ビット長であるので，33ビット以上のデータを表すには，いくつかのレジスタまたはメモリを使ってデータを表すことになる．この多倍精度の演算を行うための命令がADDX, SUBX, NEGXである．

a．ADDX (Add Multi-precision) 命令†2 この命令はデスティネーション オペランドの内容とソース オペランドの内容とXフラグを加算してデスティネーションへ格納する命令である．脚注に示すようにデータ レジスタ間の加算およびメモリ間（プリデクリメント アドレス レジスタ間接形式で指定）の加算ができる．

多倍精度加算の原理を図13.1に示す．下位の桁どうしから加算を始め，上位の桁へのキャリィは，その有無をXフラグ†1 (次ページ)に記憶しておき，上位桁の加算のときにX

図 13.1 多倍精度加算

†1 NEG 〈ea〉 0−〈ea〉 ⟶ 〈ea〉
†2 ADDX Dy, Dx Dx+Dy+X フラグ ⟶ Dx
ADDX −(Ay), −(Ax) (Ax)+(Ay)+X フラグ ⟶ (Ax)

フラグの内容も含めて加算することにより実現する．

コンディション コードは Z フラグを除いて通常の ADD 命令と同様に変化する[†2]．Z フラグは演算結果が非ゼロの場合クリアされ，ゼロの場合は変化しない．この機能を使えば多倍精度数全体の演算結果がゼロであるかないかの判定に Z フラグを使うことができる．この方法を図 13.2 に示す．まず多倍精度演算ループに入る前に Z フラグを立てておく．もし演算ループの途中で演算結果が非ゼロになると，Z フラグがクリアされるので，ループを抜けたときに多倍精度数がゼロでないことがわかる．また演算ループの途中の演算結果がすべてゼロのときには，Z フラグが立ったままでループを抜けるので，多倍精度数がゼロであることがわかる．

演算結果
0⋯⋯⋯0　⋯10010　0⋯⋯⋯0　0⋯⋯⋯0
ゼロでも変化しない　非ゼロ
Zフラグ
0 ← 0 ← 1 ← 1
多倍精度数全体がゼロでないことを示す
演算結果
0⋯⋯⋯0　0⋯⋯⋯0　0⋯⋯⋯0　0⋯⋯⋯0
Zフラグ
1 ← 1 ← 1 ← 1
多倍精度数全体がゼロであることを示す

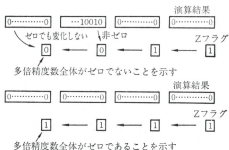



図 13.2　多倍精度演算での Z フラグの使い方

次に多倍精度加算のプログラム例を示す．図 13.2 に示すように 2 つの 4 ワード長データがそれぞれメモリに格納されている．アドレス レジスタ A0 および A1 が各先頭アドレスを指して示しているものとする．プログラムではまず A0，A1 の値を 8 増して下位桁を指し示すようにする．4 ワードの加算ができるようにループ カウンタ用のデータ レジスタ D0 に 3 をセットする．MOVE　#$04，CCR 命令で Z フラグをセットし他のフラグをクリアする．次に加算ループに入るが，このループは X フラグに変化を与えない DBRA 命令を用いて作る．

```
        ADDQ .L   #8,A0           下位桁のポインタ アドレスをセット
                                  する．
        ADDQ .L   #8,A1
        MOVEQ .L  #3,D0           ループ カウンタ セット
        MOVE      #$04,CCR        X フラグ クリア
L01     ADDX      -(A0),-(A1)     拡張加算
        DBRA      D0,L01
```

このループを抜けたとき，X, Z, N, V の各フラグはそれぞれ，多倍精度数全体に対す

†1（前ページ）多倍精度演算命令では桁上げ情報を X フラグに記憶させている．このため多倍精度演算ループではループ中の他の命令が X フラグを変化させてはいけない．DBcc 命令や CLR 命令などは，X フラグを変化させないため，ループの中に含めてよい．C フラグは多くの命令で変化するため桁上げ情報の保持には適さない．

†2　6 章参照

るキャリィ，ゼロ，負，オーバフローを示している．

b. SUBX (Subtract Multi-precision) 命令[†1]　この命令は，デスティネーションオペランドの内容からソースオペランドの内容とXフラグを減算して，デスティネーションに格納する命令である．データレジスタ間の減算およびメモリ間の減算ができる．

多倍精度数の減算は，下位の桁からのボローをXフラグに記憶しておき，次の桁の減算でXフラグを含めて減算するとにより実現される．

c. NEGX (Negate Multi-precision) 命令[†2]　この命令は，0からデスティネーションオペランドの内容とXフラグの内容を減算して，デスティネーションへ格納する命令である．この命令を使って多倍精度数全体の補数をとることができる．コンディションコードはSUBX命令と同様な変化をする．

13.1.5 クリア命令

CLR (Clear Operand) 命令[†3]　この命令は，デスティネーションオペランドの全ビットをゼロにクリアする命令である．コンディションコードに関しては，Xフラグが変化せず，Zフラグがセットされ，他のフラグはクリアされる．

13.1.6 符号拡張命令

EXT (Extend) 命令[†4]　データレジスタの符号ビットにあたるビットを上位方向にコピーすることにより，バイトデータをワードデータへ，またはワードデータをロングワードデータへ変換する命令である．この様子を図13.3に示す．オペランドサイズがワードの場合，データレジスタのビット7を8〜15ビットに拡張し，オペランドサイズがロングワードの場合，

図 13.3 符号拡張命令の操作

ビット15を16〜31ビットに拡張する．コンディションコードのXフラグは変化せず，Vフラグ，Cフラグはゼロにリセットされ，Nフラグ，Zフラグは演算結果によりセットされる．

---

†1 SUBX Dy, Dx　　Dx−Dy−X フラグ ⟶ Dx
　　SUBX −(Ay), −(Ax)　　(Ax)−(Ay)−X フラグ ⟶ (Ax)
†2 NEGX <ea>　　0−<ea>−X フラグ ⟶ <ea>
†3 CLR <ea>　　0 ⟶ <ea>
†4 EXT Dn　　ワードのとき7ビット目 ⟶ 8〜15ビット，
　　　　ロングワードのとき15ビット目 ⟶ 16〜31ビット

13.1.7 テスト命令

TST (Test) 命令[†1]　この命令は，デスティネーション オペランドが負であるかゼロであるかをコンディション コードのNフラグまたはZフラグへセットする命令である．Xフラグは影響されず，VフラグとCフラグはゼロにリセットされる．テスト命令は，オペランドが正，負，ゼロのいずれかによってブランチするための条件づくりに使われる．

13.1.8 比較命令

比較命令は，デスティネーション オペランドの内容からソース オペランドの内容を減算し，コンディション コードをセットする命令である．各オペランドの内容は比較命令の実行後にも変化しない．通常，比較命令のあとにブランチ命令を置き条件ブランチを実現する．

68000の比較命令にはCMP, CMPA, CMPI, CMPM命令の4種類がある．

a． CMP (Compare) 命令[†2]　この命令は，デスティネーション オペランドで指定したデータレジスタDnの内容から実効アドレスで指定したソース オペランドの内容を減算し，その結果に従ってコンディションコードを設定する命令である．データレジスタの内容は変化しない．実効アドレスにはすべてのアドレス形式が可能であるが，アドレス レジスタを指定するときにはバイト サイズのオペレーションはできない．

次にCMP命令を用いた最大値検索のプログラム例を示す．図13.4に示すようにアドレス レジスタA0で指し示されるメモリ領域に，検索用データが収納されているテーブルがあるとする．データはワード長のゼロでない符号なし数で，テーブルの終りにはゼロが入っているとする．初期設定としてレジスタD0をクリアする．まずテーブル内のデータをレジスタD1に取り出す．取り出した内容がゼロであればテーブルの終りなのでこのルーチンを終了する．ゼロでなければデータレジスタD1, D0の内容を比較し大きい方の値をD0にセットして次の比較へすす

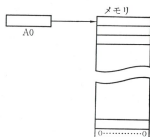



図 13.4　最大値検索プログラムのデータ

†1 TST 〈ea〉　　0−〈ea〉⟶ フラグへ反映

†2 CMP 〈ea〉, Dn　　Dn−〈ea〉⟶ フラグへ反映

む．このルーチンを終了したときには，データ レジスタ D0 に最大値を得る．

```
        CLR   D0          データ レジスタのクリア
L01     MOVE  (A0)+,D1    データを D1 へ取り出す
        BEQ   NEXT        テーブルの終りであれば NEXT へ
        CMP   D1,D0       比較
        BHI   L01         小さければそのまま次のデータへ
        MOVE  D1,D0       大きければ D0 へセット
        BRA   L01         次のデータへ
NEXT    EQU   *           NEXT STEP
```

b. CMPA (Compare Address) 命令†1　この命令は，デスティネーション オペランドで指定したアドレス レジスタ An の内容からソース オペランドの内容を減算し，結果に従ってコンディション コードを設定する命令である．アドレス レジスタの内容は変化しない．オペランド サイズはワード，ロング ワードの2種類が指定でき，ソース オペランドの実効アドレスにはすべてのアドレス形式が指定できる．

c. CMPI (Compare Immediate) 命令†2　この命令は，デスティネーション オペランドの内容からイミディエイトデータを減算し，結果に従ってコンディションコードを設定する命令である．デスティネーション オペランドの内容は変化しない．

d. CMPM (Compare Memory) 命令†3　この命令は，デスティネーション オペランドの内容からソース オペランドの内容を減算し，結果に従ってコンディションコードを設定する命令である．両オペランドの内容は変化しない．どちらのオペランドともポストインクリメント アドレス レジスタ間接形式でアドレッシングされるので，メモリの内容を順番に比較する場合に便利な命令である．

13.1.9 乗 算 命 令

乗算命令には符号付および符号なしの乗算命令がある．乗算命令は，データ レジスタの下位16ビットのデータとソース オペランド16ビットのデータとを乗算し32ビットの積をデータ レジスタに格納する命令である．

a. MULU (Unsigned Multiply) 命令†4　この命令は2つの符号なし16ビット整数を乗算し，符号なし32ビット整数を求める命令である．ソース オペランドの実効アドレ

†1 CMPA 〈ea〉, An　　An−〈ea〉 ⟶ フラグへ反映
†2 CMPI #〈データ〉, 〈ea〉　　〈ea〉− #〈データ〉 ⟶ フラグへ反映
†3 CMPM Ay@+, Ax@+　　(Ax)−(Ay) ⟶ フラグへ反映
†4 MULU 〈ea〉, Dn　　Dn×〈ea〉 ⟶ Dn

スは，アドレス レジスタ直接形式を除いて他のすべてのアドレス形式で指定することができる．特にデータ レジスタを指定するときにはそのレジスタの下位16ビットのみが取り出され，上位16ビットは無視される．デスティネーション オペランドに指定したデータレジスタには32ビットが全部格納される．

演算結果によって，コンディション コードのZフラグとNフラグがセットまたはクリアされる．2つの16ビット数の積を32ビットで表すので，オーバフローが起きることはなくVフラグおよびCフラグはクリアされる．またXフラグは影響されない．

次に2つの符号付32ビット整数を乗算し，符号付64ビット整数を求めるサブルーチンの例を示す．このプログラムは，データ レジスタD0およびD1にある2つの符号付32ビット整数の積を求め，下位32ビットをD0に，上位32ビットをD1に出力するルーチンである．MULU命令では16ビット整数どうしの積しか求められない．そこで図13.5に示すように部分積を求め，後に加算する方法をとる．プログラムではSTEP0で2つの数の符号を調べ，正の数に直し，積の符号をD7に記憶する．STEP1では部分積を求めるためデータを図13.6のようにレジスタに設定する．STEP2で各部分積を求める．部分積ADとBCのMSBはゼロなので，STEP3で和を求めたときキャリィは発生しない．STEP4で64ビットの積を求め，STEP5で符号をセットする．

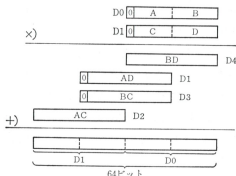



図 13.5　32ビット乗算の部分積

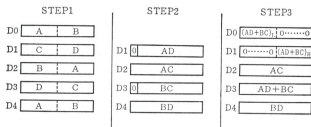



図 13.6　各ステップでのレジスタ内容

```
MUL 64   EQU      *
         CLR.W    D7
STEP 0   TST.L    D0         D0 データの符号を調べる
         BPL.S    SIGN 2
         NEG.L    D0         負なら正に直す
         NOT.W    D7
SIGN 2   TST.L    D1         D1 データの符号を調べる
         BPL.S    STEP 1
         NEG.L    D1         負なら正に直す
         NOT.W    D7         積の符号を D7 に記憶する
STEP 1   MOVE.L   D0,D2
         MOVE.L   D1,D3
         SWAP     D2
         SWAP     D3
         MOVE.L   D0,D4
STEP 2   MULU     D1,D4      部分積 BD
         MULU     D2,D1      部分積 AD
         MULU     D3,D2      部分積 AC
         MULU     D0,D3      部分積 BC
STEP 3   ADD.L    D1,D3      AD+BC
         CLR.L    D0
         CLR.L    D1
         MOVE.W   D3,D0
         SWAP     D0         (AD+BC)L,0
         SWAP     D3
         MOVE.W   D3,D1      0,(AD+BC)H
STEP 4   ADD.L    D4,D0      下位 32 ビット
         ADDX.L   D2,D1      上位 32 ビット
STEP 5   TST.W    D7
         BEQ.S    EXT
         NEG.L    D0         負のとき補数に直す
         NEGX.L   D1
EXT      RTS
         END
```

b. MULS (Signed Multiply) 命令† この命令は，2つの符号付16ビット整数を乗算し，符号付32ビット整数を求める命令である．演算は符号付の算術演算で行われる．オペランドのアドレス形式，扱うビット長およびコンディションコードの変化は，MULU 命令と等しい．

---

† MULS 〈ea〉, Dn　　Dn×〈ea〉⟶Dn

13.1.10 除 算 命 令

除算命令には符号付および符号なしの除算命令がある．除算命令はデータ レジスタの32ビットの整数をソース オペランドの16ビット整数で除算し，商と余りをデータ レジスタに格納する．図13.7に示すように結果の32ビットの内容は次のとおりである．

（1） 商は下位ワード（下位16ビット）に格納されている．

（2） 余りは上位ワード（上位16ビット）に格納されている．

D0	32ビット	データ	被除数
÷			除 数
→ D0	余 り	商	

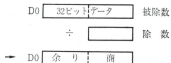



図 13.7 除算命令

符号付および符号なし除算命令はともに次の2つの特別な実行結果がある．

（1） 0で割るとトラップが生じ，プロセッサは例外処理を開始する[†1]．

（2） 命令完了前にオーバフローを検出すると，Vフラグがセットされ命令は実行されない．この場合データ レジスタの値は更新されない．

a. DIVU (Unsigned Divide) 命令[†2]　この命令はデータを符号なしの数とみなして除算する命令である．ソース オペランドはアドレス レジスタ直接形式を除いて，他のアドレス形式で指定することができる．

この除算命令はデータのビット長の関係からオーバフローが生じやすいので注意を要する．たとえばデータ レジスタの32ビット目に1が立っているとき，これを1で割ると，商は32ビット データとなり16ビットに収まりきらないためオーバフローとなる．そこで次のようなプログラムを使うことにより除算でのオーバフローを起さないようにすることができる．このプログラムでは32ビットのデータを上位16ビットと下位16ビットに分けて16ビットのデータで割って32ビットの商と16ビットの余りを得る．被除数はデータ レジスタ D0 の32ビットに，除数はデータ レジスタ D2 の下位16ビットに格納されているとする．まず被除数の下位16ビットをデータ レジスタ D1 に移し，上位16ビットをレジスタ D0 の下位側へ移す．上位16ビットのデータをデータレジスタ D2 の除数で割ると，商が D0 の下位へ，余りが D0 の上位へ入る．この除算では，16ビット データを16ビット データで割ることになるので，オーバフローが発生することはない．次にこの余りと D1 に格納された被除数の下位16ビットを加え，これを除数で割る．この除

---

†1　8章参照

†2　DIVU 〈ea〉, Dn　　Dn÷〈ea〉⟶Dn

図 13.8 オーバフローしない除算ルーチン

算でもオーバフローは発生しない．下位側の商をデータ レジスタ D0 の下位 16 ビットに移せば，D0 全体で 32 ビットの商となる．このルーチンにおけるレジスタ内容の変化を図 13.8 に示す．

```
CLR      D1
MOVE.W   D0,D1          被除数の下位16ビット
SWAP     D0
ANDI.L   #$FFFF,D0      被除数の上位16ビット
DIVU     D2,D0          除数で割る.
SWAP     D0
SWAP     D1
MOVE.W   D0,D1          余りを下位16ビットに移す
SWAP     D1
DIVU     D2,D1
MOVE.W   D1,D0          下位，上位の商を合成する
```

このルーチンを出るときに，余りはデータ レジスタ D1 の上位 16 ビットに入っている．

b. DIVS (Signed Divide) 命令†　この命令はデータを符号付の数とみなして除算する命令である．オペランドのアドレス形式，扱うビット長およびコンディション コードの変化は DIVU 命令と同じである．

余りが 0 でない場合，余りの符号は常に被除数と同じになる．たとえば，次のルーチンに示すように −12 を 5 で割ると，商は −2 となり，余りが −2 となる．

```
MOVE.L   #-12,D2        セット データ
DIVS     #5,D2          -12/5
MOVE.W   D2,QUONT       -2 (商)
SWAP     D2
MOVE.W   D2,RMNDR       -2 (余り)
```

---

† DIVS ⟨ea⟩, Dn　　Dn÷⟨ea⟩ ⟶ Dn

13.2 論理演算命令

68000の論理演算命令には論理積，論理和，排他的論理和および否定命令がある．イミディエイト論理命令と否定命令を除いた通常の論理命令では，演算は必ずデータ レジスタと他のオペランドとの間で行われる．ソース オペランドにデータ レジスタを指定した場合デスティネーション オペランドにはアドレス レジスタ直接形式およびディスプレースメント付/インデックス付プログラムカウンタ相対形式を除いた，他のアドレス形式を用いることができる．またデスティネーション オペランドにデータ レジスタを指定した場合には，ソース オペランドにアドレス レジスタ直接形式を除いた他のアドレス形式を用いることができる．アドレス レジスタ上のデータに論理演算を行うことはできない．

コンディション コードの変化は，イミディエイト論理命令でコンディション コードレジスタまたはステータス レジスタに直接論理演算を施す場合を除くと，どの論理演算でも共通で，6章の表6.2に示したようになる．この表によれば Z, N フラグが変化し，C, V フラグはクリアされる．また X フラグは前の値が保持される．

論理演算のオペランド サイズはバイト，ワード，ロングワードのすべてが可能である．イミディエイト論理演算命令（ANDI, ORI, EORI）ではデスティネーションオペランドとしてコンディションコード レジスタまたはステータス レジスタを指定できるが，これらについては特権命令にもなるので，16, 17章にて改めて説明する．

a. AND (Logical AND) 命令†　この命令は，デスティネーション オペランドの内容とソース オペランドの内容との論理積をとり，デスティネーションへ格納する命令である．

AND命令の使用例としては，ビット データの不要な部分にマスクをかけて消し，必要なビットのみを残す操作に用いる場合がある．たとえば ASCII コードから数値データを取り出す場合がそうである．数字の0から9を表す ASCII コードは16進数でそれぞれ "$ 30", "$ 31", …, "$ 39" である．そこでデータ レジスタ D0 に数字コードが入っているとき下のプログラムのようにして下位4ビットのみ取り出せば数値データに変換できる．

```
        AND.L   MASK,D0
          :
MASK:   DC.L    $000F      マスク データ
```

† AND ⟨ea⟩, Dn　　Dn·⟨ea⟩ ⟶ Dn
　AND Dn, ⟨ea⟩　　⟨ea⟩·Dn ⟶ ⟨ea⟩

b. ANDI (AND Immediate) 命令[†1]　この命令は，デスティネーション オペランドの内容とイミディエイト データとの論理積をとり，デスティネーションへ格納する命令である．

AND 命令で示したプログラムを ANDI 命令で書き直すと

```
ANDI .L    #$0F,D0
```

となり，マスク情報が命令中に含まれるためわかりやすくなる．

c. OR (Inclusive OR logical) 命令[†2]　この命令は，デスティネーション オペランドの内容とソース オペランドの内容の論理和をとり，デスティネーションへ格納する命令である．

OR 命令はあるビット位置に無条件に 1 を立てるときや，データの一部に他のデータをはめこむときに用いることもできる．後者の一例として，ASCII 文字データで 2 進化 10 進数の 2 桁がデータ レジスタ D0 と D1 に入っているとき，データ レジスタ D0 に 2 進化 10 進数値をパックするルーチンは以下のようになる．

```
ANDI .L    #$0F,D0
LSL .W     #4,D1
OR .B      D1,D0
```

d. ORI (Inclusive OR Immediate) 命令[†3]　この命令は，デスティネーションオペランドの内容とイミディエイト データとの論理和をとり，デスティネーションへ格納する命令である．

e. EOR (Exclusive OR Logical) 命令[†4]　この命令は，デスティネーション オペランドの内容とソース オペランドの内容との排他的論理和をとり，デスティネーションへ格納する命令である．EOR 命令では AND 命令，OR 命令と異なり，ソース オペランドがデータ レジスタのみに限定されている．

f. EORI (Exclusive OR Immediate) 命令[†5]　この命令は，デスティネーション オペランドの内容とイミディエイト データとの排他的論理和をとり，デスティネーションへ格納する命令である．

---

†1 ANDI #〈データ〉, 〈ea〉　〈ea〉·#〈データ〉 ⟶ 〈ea〉
†2 OR 〈ea〉, Dn　Dn∨〈ea〉 ⟶ Dn
　OR Dn, 〈ea〉　〈ea〉∨Dn ⟶ 〈ea〉
†3 ORI #〈データ〉, 〈ea〉　〈ea〉∨#〈データ〉 ⟶ 〈ea〉
†4 EOR Dn, 〈ea〉　〈ea〉⊕Dn ⟶ 〈ea〉
†5 EORI #〈データ〉, 〈ea〉　〈ea〉⊕# データ ⟶ 〈ea〉

g. NOT (Logical Complement) 命令†1　この命令は，デスティネーション オペランドの内容の1の補数をとり（各ビットを反転させ），デスティネーションへ格納する命令である．

13.3　2進化10進数演算命令

68000には2進化10進数を演算する加算，減算および補数命令がある．これらの命令は1バイトにつめられた2進化10進数 BCD (Binary Coded Decimal) の2桁を単位として演算を行う．そして多数桁の2進化10進数演算ができるように拡張フラグ(Xフラグ)を含めた演算を実行する．

a. ABCD (Add Digits) 命令†2　この命令は，Xフラグを含めてバイト長のオペランドどうしの2進化10進数加算を行う命令である．オペランドの指定のしかたは2とおりの方法がある．

（1）データ レジスタ間　　ABCD　　Dy, Dx

（2）メモリ間　　　　　　ABCD　　−(Ay), −(Ax)

メモリ間の演算では，どちらのオペランドもプリトデクリメント アドレス レジスタ間接形式で指定する．またオペレーション サイズはバイト長だけが許されている．本命令ではコンディション コードのNフラグおよびVフラグは，意味が無いため定義されない．CフラグおよびXフラグは10進加算でキャリィが発生したときセットされ，その他のときクリアされる．Zフラグは演算結果がゼロでないときクリアされ，その他のときは変化しない．したがって多倍精度2進数のときと同じように，多バイトで表現された2進化10進数全体の演算結果がゼロであるかを判別することができる．

次に4バイト2進化10進数加算のプログラムを示す．図13.9に示すように，2つの4バイト2進化10進数がメモリに格納されており，ルーチンに入る前に，アドレス レジスタA0, A1がそれぞれのデータの先頭アドレスを指し示しているとする．このプログラムではまず各アドレス レジスタを4番地増加し最下位桁の1つ先のバイトを指し示させる．次に

A0 →

10^7	10^6
10^5	10^4
10^3	10^2
10^1	10^0位

A1 →　　Byte

10^7	10^6
10^5	10^4
10^3	10^2
10^1	10^0位

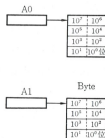



図 13.9　2進化10進数加算のデータ構造

†1　NOT　⟨ea⟩　　　$\overline{\langle ea \rangle} \longrightarrow \langle ea \rangle$

†2　ABCD　Dy, Dx　　　Dx+Dy+X フラグ ⟶ Dx

　　ABCD　−(Ay)−(Ax)　　(Ax)+(Ay)+X フラグ ⟶ (Ax)

ループカウンタに使うレジスタ D0 に 3 をセットし，X フラグをクリアし，Z フラグをセットする．この後 BCD 加算のループに入り 4 バイトの和データを得る．

```
        ADDQ .L    #4,A0             下位バイト ポイント
        ADDQ .L    #4,A1             下位バイト ポイント
        MOVEQ .L   #3,D0             ループ カウンタ セット
        MOVE       #$04,CCR          CCR クリア
L01     ABCD       -(A0),-(A1)       10進加算
        DBRA       D0,L01
```

このルーチンを出たあと，Z フラグは演算結果の 4 バイト 2 進化 10 進数がゼロであるかどうかを示しており，C フラグは 4 バイト 2 進化 10 進数の加算でのキャリィを示している．

b. **SBCD (Subtract Digits) 命令**†　この命令は，X フラグを下位桁からのボローとして，2 進化 10 進数の減算を行う命令である．オペランドの指定の方法およびオペレーション サイズは ABCD 命令と変らない．コンディション コードは，C フラグおよび X フラグが 10 進数減算でのボローが発生したときに，セットされ，その他のときにクリアされる．他のフラグは ABCD 命令と同じ変化をする．

次のプログラムは図 13.9 に示される 4 バイト 2 進化 10 進数の減算を行うルーチンである．

```
        ADDQ .L    #4,A0             下位バイト ポイント
        ADDQ .L    #4,A1             下位バイト ポイント
        MOVEQ .L   #3,D0             ループ カウンタ セット
        MOVE       #$04,CCR          CCR, Z フラグ セット
L01     SBCD       -(A0),-(A1)       10進減算
        DBRA       D0,L01
```

このルーチンを出たあと，Z フラグは結果の 4 バイト 2 進化 10 進数がゼロであるかどうかを示している．C フラグは結果が真数であるか，負の数の補数表示であるかを示して

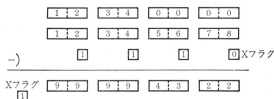



図 13.10　2進化10進数減算の数値例

† SBCD　Dy, Dx　　　Dx−Dy ⟶ Dx
　SBCD　−(Ay), −(Ax)　(Ax)−(Ay) ⟶ (Ax)

いる．図13.10にこのような減算の数値例を示す．この補数表示を真数表示に直すには次の NBCD 命令を使えばよい．

c．NBCD (Negate Digit) 命令[†1]　この命令は2進化10進数の10の補数または9の補数をとる命令である．この命令では，デスティネーション オペランドの内容とXフラグをゼロから引いて，デスティネーションに格納する．したがってXフラグがクリアされているときには10の補数が得られ，またXフラグが立っているときには9の補数が得られる．オペレーション サイズはバイト長だけが許されている．デスティネーション オペランドのアドレス形式は，アドレス レジスタとディスプレースメント付/インデックス付プログラム カウンタ相対形式を除いて他のすべてが可能である．コンディション コードは SBCD 命令と同様の変化をする．

13.4 テスト アンド セット命令

TAS (Test and Set Operand) 命令[†2]　テスト アンド セット命令は，オペランドの値をテストし，同じバスサイクルでオペランドのビットをセットする命令である．この命令の動作を図13.11に示す．この命令はマルチプロセッサ システムで共通メモリを誤りなく参照するために便利である．

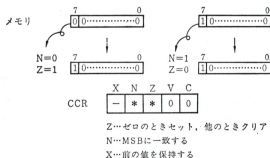



図 13.11　TAS 命令の動作

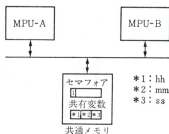
オペランド サイズはバイトのみであり，テストの結果はコンディション コードのNフラグまたはZフラグに反映される．

次にマルチプロセッサ システムで共通メモリを参照する方法を説明する．図13.12に示

---

†1　NBCD 〈ea〉　0−〈ea〉⟶〈ea〉

†2　TAS 〈ea〉　〈ea〉テスト ⟶ cc；〈ea〉の第7ビットに1をセット

すように2つのプロセッサMPU-AとMPU-Bが共通メモリをもっており，このメモリ中に2つのプロセッサが共有する変数があるとする．共有変数として時刻を表す変数hh, mm, ssを考えよう．MPU-Aは外部からクロックの割込みにより時刻変数を更新し，またMPU-Bは必要に応じて時刻変数を参照する．

いま時刻が10時59分59秒とすると時刻変数の値はhh=10，mm=59，ss=59となっている．外部からの割込みがあるとMPU-Aは時刻変数の値を更新しはじめる．ss=00，mm=00まで更新したときに，たまたまMPU-Bが時刻変数を参照すると，10時00分00秒と読み取られ，本当の時刻11時00分00秒とは異なり，誤りとなる．

MPU-A　MPU-B
セマフォア
1
共有変数
*1 *2 *3
共通メモリ
*1：hh
*2：mm
*3：ss

図 13.12 マルチプロセッサ システム

誤りを防止するには共有変数を同時に2つのプロセッサからアクセスしないようにすればよい．これを相互排除（Mutual Exclusion）の原理という．

ソフトウェアで相互排除を行うには，共有変数の参照状態を示すセマフォア†とよばれるフラグを用いる．セマフォアの値が1であるときはそのセマフォアに対応する共有変数がどれかのプロセスから参照されていることを示す．各プロセッサは共有変数を参照するときにはセマフォアを調べ，1であれば他のプロセッサの参照が終るまでまつ．0であればすぐに1を立てて他のプロセッサに共有変数が参照中であることを示す．参照を終了するときにはセマフォアを0に戻さなければならない．セマフォアを調べるときTAS命令の代りにテスト命令とセット命令を用いると相互排除ができない．なぜなら一方のプロセッサのテスト命令とセット命令の実行の間に，他方のプロセッサが割り込んで同時に共有変数の参照に入る可能性があるからである．

以上から共有変数を参照するルーチンの例は次のように書ける．

```
L01:  TAS        SEM    セマフォアの値をCCRに反映し，1を立てる
      BMI.S      L01    1であればループして待つ
      共有変数の参照      0であれば共有変数の参照に入る
            ≀           このときセマフォアは1になっている
      CLR        SEM    セマフォアを0にして参照を終了する
```

† セマフォアとは鉄道の腕木式信号機を意味する言葉である．

14　桁移動，ビット操作命令

この章ではレジスタやメモリの内容を桁移動したり，桁送りしたりする命令およびレジスタやメモリの特定ビットを判定したり，操作を加えたりする命令について述べる．前者はシフト，ローテート命令グループ，後者はビット操作命令グループに分けられる．68000では4つのシフト命令（ASL, ASR, LSL, LSR 命令），4つのローテート命令（ROL, ROR, ROXL, ROXR 命令）および4つのビット操作命令（BTST, BSET, BCLR, BCHG 命令）が用意されている．これらの命令はバイト，ワード，ロングワードのオペランドサイズや豊富なアドレス形式を組み合せることができ，多くの操作を容易に実現できる．

14.1　論理形桁移動（論理シフト）命令

LSL, LSR (Logical Shift Left/Right) 命令†　レジスタやメモリの内容を桁移動させる操作が論理シフトである．68000には2つの論理シフト命令 LSL, LSR 命令が用意されている．これらはデスティネーションオペランドのバイト，ワードあるいはロングワードのデータを単純に左へ（LSL）または右へ（LSR）動かすものである．このときシフトされてオペランドの外へ出てしまったビットは順次コンディションコードレジスタ内にあるCフラグおよびXフラグに移り，もとのオペランドレジスタからは捨てられてしまう．逆の端のビット位置には0が入る．68000の論理形桁移動命令はデータレジスタの内容をシフトする場合とメモリの内容をシフトする場合とがあり――算術形桁移動命令，循環形桁送り命令も同様――両者の機能が若干異なる．

---

† LSL/LSR	#〈データ〉, Dy	Dy の内容を #〈データ〉だけシフト
LSL/LSR	Dx, Dy	Dy の内容を Dx の内容だけシフト
LSL/LSR	〈ea〉	〈ea〉の内容を1ビットだけシフト

まず，データ レジスタの内容のシフトについて述べる．この場合オペランド サイズはバイト，ワード，ロング ワードが許されている．データ レジスタの内容をシフトするためには，ソース オペランドで桁移動の数を指定し，デスティネーション オペランドで被シフト データ レジスタを指定する．ソース オペランドの指定，すなわち桁移動の数の指定は次の2とおりの方法による．

（1） イミディエイト データ（スタティック）　シフト カウントをイミディエイト データで直接指定する．シフト カウントは1～8の範囲に限られる．

（2） データ レジスタ（ダイナミック）　シフト カウントをデータ レジスタにセットし，そのデータ レジスタを指定する．シフトカウントはそのデータ レジスタの下位6ビットに格納された値が有効となり，上位ビットは無視される．したがってシフト カウントは0～63の範囲となる．たとえばデータ レジスタに67が格納されていたとすると，これは2進数，01000011であるから下位6ビット分の値は000011，すなわち3がシフト カウントになる．

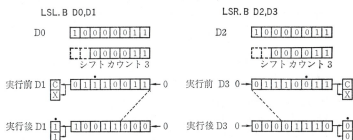



図 14.1　論理シフト命令 LSL，LSR の例

図14.1は左論理シフト命令 LSL および 右論理シフト命令 LSR でデータ レジスタ D1 に格納された $73 を左右論理シフトした例である．シフト カウントはデータ レジスタ D0 で指定する．データ レジスタには16進数 $83 が格納されているが，6ビット（$2^6-1=63$）分のデータのみが有効となりシフト カウントは3となる．LSL 命令によってオペランドの最上位ビットからシフトされて出てしまったビットは C フラグおよび X フラグに移され，LSR 命令では最下位ビットからシフトされて外へ出たビットが C フラグおよび X フラグに順次移されるので，命令実行後 LSL 命令の例では1が，LSR 命令の例では0が C フラグおよび X フラグに格納されている．LSL の場合の“1”は命令実行前のデータレジスタ D0 の第6ビット（2^6）の“1”を反映し，LSR の場合の“0”は命令実行前の

データレジスタ D3 の第3ビット（2^3）の “0” を反映したものである．C フラグおよび X フラグはこのように最後にシフトされて外に出たビットの値が格納される．ただし，シフトカウントが0のときは，C フラグは常に0となり，X フラグは変化せず前の値が保存されるようになっている．

シフト カウントが1～8の範囲においては，イミディエイト データによって直接シフトカウントを指定した方が便利である．データ レジスタにシフト カウントを設定する必要がなく，単一命令で桁移動を実行できるからプログラムのステップ数が小さくでき，処理時間も短縮できる．

次に，メモリの内容をシフトする場合について述べる．シフト カウントは常に1ビットであり，オペランド サイズもワード だけに限られる．したがってこの場合はシフト カウントやオペランド サイズを指定することはできない．これらを指定して論理シフトしたい場合はメモリ上のデータを一旦データ レジスタに取り出したうえで前述の論理シフトを行えばよい．

論理形桁移動は符号なし整数の乗・除算に用いることができる．また各種のデータ中の特定の部分を操作してデータを加工したり，並べ換えたりするのにも用いられる．次のルーチンは論理シフトを用いて乗算を行う例で，符号なしの16ビット（ワード）データを10倍するものである．乗算結果が符号なしの16ビットに収まるには被乗数は6553以下である必要がある．

```
MULT   MOVE.W   DATA1,D0     被乗数取出し
       LSL.W    #1,D0        被乗数×2
       MOVE.W   D0,D1        退避
       LSL.W    #2,D0        被乗数×8
       ADD.W    D1,D0        被乗数×(2+8)=被乗数×10
```

LSL 命令により1ビット左へ論理シフトすることは2倍することに相当し，n ビット左へ論理シフトすることは 2^n 倍することに相当する．また1ビット右へ論理シフトすることは1/2倍，すなわち2で割ることに相当し，n ビット右へ論理シフトすることは $1/2^n$ 倍，すなわち 2^n で割ることに等価である．次の例は符号なしの16ビット（ワード）データを8で割るルーチンである．除算結果は小数点以下が切り捨てられる．

```
DIVD   MOVE.W   DATA1,D0     被除数取出し
       LSR.W    #3,D0        被除数÷8
```

メモリ上の16ビット（ワード）データを直接8で割るには，1ビットの右論理シフトを3回繰り返せばよい．この場合はデータ レジスタを介さないで除算が実行される．

```
DIVM    LSR.W    DATA1         被除数÷2
        LSR.W    DATA1         被除数÷2
        LSR.W    DATA1         被除数÷2
```

除算結果はもとのメモリ上の同一番地 DATA1 に与えられるため，被除数は退避しておかないかぎり失われてしまう．

次のルーチンはメモリ上の8ビット（バイト）データを2桁の16進数表示で外部装置に出力するため，ASCII 符号に変換するものである．論理シフトデータの一部が処理に用いられている．

```
BIAS    MOVE.B   DATA1,D1      16進数の2桁の取出し（上位桁）
        LSR.B    #4,D1         下位4ビットを消す
        ORI.B    #$30,D1       ASCII に変換
        CMPI.B   #$3A,D1       A~F 補正か？
        BLT.S    BS1           補正不要なら分岐
        ADDI.B   #7,D1         A~F 補正
BS1     MOVE.B   D1,BUF1       変換結果の退避（上位桁）
        MOVE.B   DATA1,D1      16進数2桁の取出し（下位桁）
        LSL.B    #4,D1         上位ビットを消す
        LSR.B    #4,D1         変換準備
        ORI.B    #$30,D1       ASCII に変換
        CMPI.B   #$3A,D1       A~F 補正か
        BLT.S    BS2           補正不要なら分岐
        ADDI.B   #7,D1         A~F 補正
BS2     MOVE.B   D1,BUF2       変換結果の退避（下位桁）
```

以上により DATA1 に格納されていた8ビット（バイト）データ2桁の16進数表示の ASCII 符号0~9，A~F に変換され，BUF1 と BUF2 に出力される．

逆に16進数表示のデータの ASCII 符号が与えられたとき，これを2進データに変換するルーチンは論理シフトを用いて同様にプログラミングすることができる．これについては19章のプログラム例を参照されたい．

14.2 算術形桁移動（算術シフト）命令

ASL, ASR (Arithmetic Shift Left/Right) 命令† 算術形桁移動は2の補数表示さ

---

† ASL/ASR #〈データ〉, Dy Dy の内容を #〈データ〉だけシフト
ASL/ASR Dx, Dy Dy の内容を Dx の内容だけシフト
ASL/ASR 〈ea〉 〈ea〉の内容を1ビットだけシフト

れた2進数を算術的，すなわち符号を保ったまま桁移動できることが特徴であり，この点が単純にオペランドを左右に桁移動する論理形桁移動と異なっている．68000では2つの算術シフト命令，すなわち左算術シフト ASL 命令と右算術シフト ASR 命令とが用意されている．ASL 命令は論理シフト LSL 命令と同様にオペランドを左へシフトするもので，左端の最上位ビットから外へ出されたビットは順次 C フラグおよび X フラグに移される．LSL 命令との違いは，シフトの際に符号の変化が生じた場合，コンディション コードの V フラグが1に設定されることである．これによりシフト データの符号の変化を知ることができる．ASR 命令は論理シフト LSR 命令と同様にオペランドを右へシフトするもので，右端の最下位ビットから外へ出されたビットは順次 C フラグおよび X フラグに移される．このとき最上位ビットすなわち符号ビットは不変でありこれが順次右へシフトされていく．なお ASR 命令では符号ビットは不変であるから，V フラグは常に0である．ASL/ASR 命令は LSL/LSR 命令と同じくデータ レジスタの内容をシフトする場合とメモリの内容をシフトする場合とがある．前者の場合，オペレーション サイズはバイト，ワード，ロング ワードが許されており，シフト カウントの指定方法も LSL/LSR 命令と同じく (1) イミディエイト データ(スタティック形) で指定する方法と，(2) データレジスタ (ダイナミック形) で指定する方法とがある．また後者の場合は，オペレーション サイズがワードのみに限られ，シフト カウントも1ビットのみである．

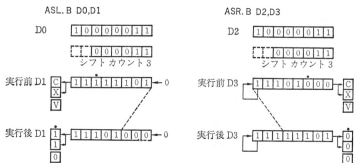



図 14.2　算術シフト命令 ASL，ASR の例

図14.2にデータ レジスタでシフト カウントを指定し，符号ビット付の8ビットデータ，−3，−24を左右算術シフトした例を示す．16ビット データ，−3，−24は16進数表示の $FD，$E8 にそれぞれに相当する．左算術シフト命令 ASL では $FD，すなわち −3 を左へ3ビット シフトして −24を得ている．これは −3 が8倍された結果である．V フラグは0で符号ビットの変化はなく，得られた値は算術シフトによる正しい結果を示してい

る．Vフラグが1になるような算術シフトにおいては，一般に正しい結果が得られていないので注意が必要である．一方右算術シフト命令 ASR では $E8, すなわち −24 を右へ3ビット シフトとして −3 を得ている．これは −24 が 1/8 倍された結果である．ASR 命令では本来の算術シフトによる結果が常に得られると考えてよい．シフト実行前のオペランドデータ，すなわち被除数が負のときは符号ビットは1に保たれ，右シフトによって空いた上位ビットには1が埋められる．被乗数が正のときは符号ビットは0に保たれ，空いた上位ビットには0が埋められる． この結果nビット右算術シフトによって符号が変ることはなく，2^n で除算された結果がそのままレジスタに得られるわけである．ただし小数点以下はレジスタの外へ棄てられてしまう．正数の算術形右シフトでは除算の商をレジスタに残し， 余りは棄てられる． たとえば 1101_2 を3ビット右シフトすると 0001_2 となり13を8で割った商は1ということになる．したがって端数である5は失われる．

負数の算術形右シフトでは様子が異なる．たとえば 11110011_2(−13) を3ビット右シフトすると，11111110_2 (−2) となり，−13を8で割った商は負数 −2 で， 端数は +3 ということになる．

算術シフトを用いると2の補数表示されたデータの簡単な乗除算を，正負の符号に拘らず，同一の手順で実行できる．次のルーチンは2の補数表示による16ビット（ワード）データを2.5倍するものである．右算術シフトで与えられたデータを2を割ったものと，左算術シフトで2倍したものとを加算して結果を得ている．

```
ASMD    MOVE.W   DATA1,D0     データ取出し
        ASL.W    #1,D0        データ×2
        MOVE.W   D0,D1        退避
        ASR.W    #2,D0        データ÷2
        ADD.W    D0,D1        2.5倍の実行
        MOVE.W   D1,DATA2     結果の退避
```

左算術シフト命令 ASL で1ビット シフトしたデータ レジスタ D0 を右算術シフト命令 ASR で2ビット シフトすると，元のデータを2で割った値が得られる．上の例で ASL 命令と ASR 命令を入れ替えることもできるが， ASR 命令をさきに実行すると小数点以下が失われるので数値計算上好ましくない．

14.3　循環形桁送り（ローテート）命令

ROL，ROR（Rotate without X Left/Right）†1，ROXL，ROXR（Rotate through X Left/Right）†2 命令　循環形桁送りはオペランドの左端と右端，すなわち最上位ビットと最下位ビットを連結して環状にし，この中でオペランドの桁送りを行うものである．オペランドのデータはこの環の中を回転（ローテート）し，一巡すると元の位置に戻る．論理形桁移動でレジスタの外に出されたビットが反対側の端から順次レジスタに入ってくると考えてもよい．68000では4つのローテート命令がある．ROL命令とROR命令はオペランドを左右にそれぞれローテートする命令である．ROXL命令とROXR命令はエクステンド ビット付ローテートとよばれ，オペランドの最上位ビットと最下位ビットの間に1ビットのXフラグを介在させ，この拡張ビットを含んだ環の中でオペランドの左右桁送りをそれぞれ行うものである．

ローテート命令においても，論理シフト命令，算術シフト命令と同様にデータ レジスタを扱う場合とメモリを扱う場合がある．データ レジスタの内容をローテートする場合には，オペレーション サイズはバイト，ワード，ロング ワードが許される．ローテートカウントは論理シフト，算術シフトの場合と同じく（1）イミディエイト データ（スタティック形）と（2）データ レジスタ（ダイナミック形）により指定することができる．一方，メモリの内容をローテートする場合，オペランド サイズは16ビット（ワード）に限られ，ローテートカウントも1ビットに限定される．メモリ上のデータを直接nビット分ローテートしたいときは，1ビットのローテート命令をn回繰り返して用いればよい．

図14.3に4つのローテート命令ROL，ROR，ROXL，ROXRによって循環形桁送りを行った例を示す．ROL（ROR）命令において最上位ビット（最下位ビット）からローテートされて外へ出てしまったビットはCフラグに入るとともに，反対端の最下位ビット（最上位ビット）に戻る．したがって命令実行後Cフラグには一番最後にローテートにより外へ

---

†1	ROL/ROR　Dx, Dy	Dyの内容をDxの内容だけローテート
	ROL/ROR　#<データ>, Dy	Dyの内容を#<データ>だけローテート
	ROL/ROR　<ea>	<ea>の内容を1ビットだけローテート
†2	ROXL/ROXR　Dx, Dy	Dyの内容をXフラグとともにDxの内容だけローテート
	ROXL/ROXR　#<データ>, Dy	Dyの内容をXフラグとともに#<データ>だけローテート
	ROXL/ROXR　<ea>	<ea>の内容をXフラグとともに1ビットだけローテート

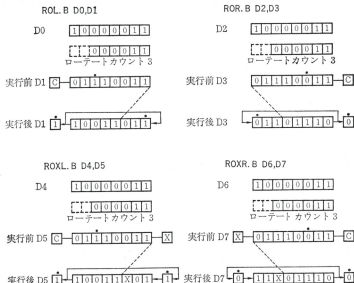



図 14.3　ローテート命令 ROL, ROR, ROXL, ROXR の例

出たビットの値が入っている．ただしローテート カウントが 0 のときは常に 0 となる．X フラグは命令実行前の値が保持される．ROXL（ROXR）命令においては最上位ビット（最下位ビット）からローテートされて外へ出てしまったビットは C フラグと X フラグに入る．また反対端の最下位ビット（最上位ビット）には X フラグの前の値がつめられる．命令実行後 C フラグ，X フラグには一番最後に外へ出されたビットの値が入っているが，ローテート カウントが 0 のときは，C フラグには命令実行前の X フラグの値が入り，X フラグの値は変化しない．

循環形桁送り命令によって 16 ビット データのパリティ ビットを求める方法を次に示す．

```
PRTY1   MOVE.W   #15,D0     回数を設定
        CLR.B    D1         パリティ ビット クリア
PR2     ROL      DATA1      1ビット ローテート
        BCC      PR1        1でなければ分岐
        EORI.B   #1,D1      パリティビット反転
PR1     DBRA     D0,PR2     16回完了でなければ分岐
```

以上により DATA1 の “1” をとるビットの数が偶数ならばパリティ ビットは 0 となり，DATA1 の “1” をとるビットの数が奇数ならばパリティ ビットは 1 となる．この値

はデータ レジスタ D1 の最下位ビットに得られる．循環形桁送りによって DATA1 のデータは破壊されないで残る．上のルーチンが終了したとき，DATA1 の値は最初の値に戻っている．

これまで述べてきた論理形，算術形シフト，およびローテートの各命令とその動作の概要は5章の表5.5にまとめてある．

14.4 ビット操作命令

ビット操作命令はレジスタやメモリ内のデータの特定のビットの状態を調べたり，特定ビットの操作を行う命令である．68000 には BTST, BSET, BCLR, BCHG の4種の命令が用意されている．

a. BTST (Bit Test) 命令[†1]　BTST 命令はオペランドのビット群のうち，指定されたビットの状態を調べ，その結果をZフラグに反映する．操作対象ビットの値が0であればZフラグは1となり，操作対象ビットが1であればZフラグは0となる．操作対象となるオペランドのサイズはデスティネーション オペランドに依存して決定され，ユーザが指定することはできない．すなわち，オペランドがデータ レジスタの場合はロング ワードとなり，指定されるビット番号は0〜31が用いられる．オペランドがメモリ内のデータの場合はバイト サイズとなり，指定されるビット番号は0〜7が用いられる．

ビット番号の指定には次の2とおりの方法がある．

(i) イミディエイト データ（スタティック）　ビット番号をイミディエイト データで直接指定する．オペランドがデータ レジスタの場合は下位5ビットのデータが有効となる．すなわち (0〜31) が用いられる．メモリ上のデータの場合は下位3ビットのデータが有効となる．すなわち (0〜7) が用いられる．

(ii) データ レジスタ（ダイナミック）　ビット番号をあらかじめデータ レジスタに設定しておき，そのデータ レジスタを指定することにより操作対象となるビットを指定する．オペランドがデータ レジスタの場合は下位5ビットのデータが有効となる．メモリ上のデータの場合は下位3ビットのデータが有効となる．

b. BSET (Bit Test and Set) 命令[†2]　BSET 命令はオペランドのビット群のうち指

---

†1	BTST	Dn, 〈ea〉	〈ea〉のビットのうち Dn で示されるビットをテスト
		#〈データ〉, 〈ea〉	〈ea〉のビットのうち #〈データ〉番目のビットをテスト
†2	BSET	Dn, 〈ea〉	〈ea〉のビットのうち Dn で示されるビットを1にセット
	BSET	#〈データ〉, 〈ea〉	〈ea〉のビットのうち #〈データ〉番目のビットを1にセット

定された番号の1ビットの状態を調べ，その結果をZフラグに反映した後，この指定したビットを1にする命令である．すなわち指定したビットが1であれば0をZフラグに設定し，指定したビットが0であれば1をZフラグに設定する．このあとで指定したビットを1にする．オペランド サイズはBTST命令と同じくデータ レジスタの場合32ビット（ロング ワード），主記憶内のデータでは8ビット（バイト）が自動的に決められる．ビット番号の指定も（1）イミディエイト データ（スタティック形）と（2）データ レジスタ（ダイナミック形）の2とおりがある．

c．BCLR (Bit Test and Clear) 命令†1　BCLR命令はオペランドのビット群のうち指定された1ビットの状態を調べて，この結果をZフラグに反映したのち，この指定ビットを0にする命令である．オペランド サイズとビット番号の指定についてはBTST命令の場合と同様である．

d．BCHG (Bit Test and Change) 命令†2　BCHG命令はオペランドのビット群のうち指定された1ビットの状態を調べて，これをZフラグに反映した後は，指定したビットを反転する命令である．すなわち指定したビットが0ならばこれを1に逆転し，1ならばこれを0に逆転するものである．この場合オペランドの他のビットは不変である．オペランドサイズ とビット番号の指定についてはBTST命令の場合と同様である．4種のビット操作命令とその動作の概要はすでに表5.6にまとめてある．

16ビット データのパリティ ビットを求めることは前述の循環形桁送り命令のほか，ビット操作命令によっても実現できる．次にそのルーチンを示す．この例では，データ レジスタD0をループ カウンタとビット位置指定用レジスタとして兼用している．

```
PRTY 2   MOVE.W   #15,D0       回数を設定
         CLR.B    D1           パリティ ビットクリア
         MOVE.W   DATA 1,D2    データ取出し
PR 2     BTST     D0,D2        K ビットをテスト (K=(D0))
         BEQ      PR 1         1でなければ分岐
         EORI.B   #1,P1        パリティ ビット反転
PR 1     DBRA     D0,PR 2      16回完了でなければ分岐
```

---

†1	BCLR Dn, <ea>	<ea>のビットのうちDnで示されるビットをテストした後，0にリセット
	BCLR #<データ>, <ea>	<ea>のビットのうち #<データ> 番目のビットをテストした後，0にリセット
†2	BCHG Dn, <ea>	<ea>のビットのうちDnで示されるビットをテストした後，このビットを反転
	BCHG #<データ>, <ea>	<ea>のビットのうち #<データ> 番目のビットをテストした後，このビットを反転

以上により DATA1 の 16 ビットのうち "1" の値をとるビットの数が偶数ならばパリティビットは 0 となり，"1" の値をとるビットの数が奇数ならばパリティ ビット は 1 となる．この値はデータ レジスタ D1 に得られる．主記憶内の 8 ビット (バイト) のデータのパリティを求める場合は BTST 命令で直接各ビットの "0"，"1" をテストできるので，データをレジスタに取り出して各ビットをテストしなくてすむ．

15 プログラム制御命令

この章ではプログラム制御命令，すなわち分岐命令，サブルーチン操作命令および条件セット命令について説明する．同時にサブルーチン操作におけるパラメータ授受の代表的な方法についても述べる．また，LINK/UNLK 命令は本来プログラム制御命令ではなくデータ転送命令に分類されるべきものであるが，サブルーチン操作と密接なかかわりあいがあるため本章で説明する．

15.1 分岐命令

68000 の分岐命令には JMP, Bcc, BRA, DBcc 命令の 4 種類がある．この中で JMP, BRA 命令は無条件分岐命令であり，Bcc, DBcc 命令は条件付ブランチ命令である．以下，各命令について説明する．

a. JMP (Jump) 命令† JMP 命令は無条件ジャンプ命令であり，分岐先アドレスの指定方法は 6 種類のアドレス形式を用いることができる．すなわちアドレス レジスタ間接形式，ディスプレースメント付アドレス レジスタ間接形式，インデックス付アドレス レジスタ間接形式，長または短絶対アドレス形式，ディスプレースメント付/インデックス付プログラムカウンタ相対形式を使用できる．

また，JMP 命令はディスプレースメント付プログラム カウンタ相対形式を除いて，アドレス空間全体に分岐することが可能である．ディスプレースメント付プログラム カウンタ相対形式は 16 ビットの符号付整数で表せる範囲 ($PC-2^{15}$～$PC+2^{15}-1$) に分岐することが可能である．

---

† JMP 〈ea〉 デスティネーション ⟶ PC

b. Bcc (Branch Conditionally)[†1], BRA (Branch Always)[†2] 命令　Bcc 命令は条件付ブランチ命令であり，条件を "cc" で表す．Bcc 命令で使用できる条件は常に真 (T)，常に偽 (F) を除く 14 種類である（5章，表5.7参照）．たとえば，「等しい場合に分岐する」という条件を示す命令は BEQ (Branch Equal) と書く．

BRA 命令は無条件ブランチ命令であり，指定した番地へ無条件に分岐する．

Bcc, BRA 命令はディスプレースメント付/インデックス付プログラム カウンタ相対形式のみ使用できる．アセンブラで記述する場合は分岐先をラベルで指定する．アセンブラは分岐命令の相対距離を計算し，その結果をディスプレースメントとする．ディスプレースメントは 8 ビットと 16 ビットの 2 とおりがある．これは分岐する範囲によって決る．すなわち，分岐命令とラベルの相対距離が $-2^7 \sim 2^7-1$ バイト以内の場合はディスプレースメントは 8 ビットでよく，分岐命令とラベルの相対距離が $-2^{16} \sim 2^{16}-1$ バイト以内の場合にはディスプレースメントは 16 ビットになる．Bcc, BRA 命令の命令長はディスプレースメントが 8 ビットの場合に 1 ワード命令になり，16 ビットの場合に 2 ワード命令になる．これをアセンブラ レベルで指定する場合には，ニモニックの後に ".S"（ショートの意味）を付けることで区別する．すなわち，

```
BEQ.S   〈ラベル〉
```

は 1 ワードのオブジェクト コードを生成し，

```
BEQ     〈ラベル〉
```

は 2 ワードのオブジェクト コードを生成する．

次に Bcc 命令の使用例を示す．これは CMP 命令でデータ レジスタ D0 と D1 を比較してコンディション コードに結果を設定し，Bcc 命令で条件をテストするものである．

```
CMP     D0,D1
BEQ     〈ラベル〉
```

また，Bcc 命令は ADD, SUB, MULS, DIVS 命令などの直後に置き，各演算の結果により処理を選択するときにも用いられる．

JMP, Bcc, BRA 命令の比較表を表 15.1 に示す．この表より，JMP 命令はアドレス空間全体に分岐できるので，JMP 命令と飛び先との相対距離が大きい場合に利用できる．逆に BRA 命令は相対距離が小さい場合に利用できる．さらに，Bcc 命令はテスト コンディションを用いた条件分岐に利用できる．

---

†1　Bcc　〈ラベル〉　cc が真ならば〈ラベル〉へ分岐，偽ならば次の命令を実行
†2　BRA　〈ラベル〉　無条件に〈ラベル〉へ分岐

表 15.1　JMP, Bcc, BRA 命令比較表

	命令	JMP 命令	Bcc 命令	BRA 命令
分岐する範囲	PC−128〜PC+127バイト	1〜2 ワード	1 ワード	1 ワード
	PC−32 768〜PC+32 767	1〜2 ワード	2 ワード	2 ワード
	アドレス空間全体	1〜3 ワード	×	×
条件分岐		×	○	×

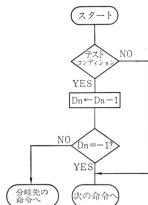
c. DBcc (Decrement Counter and Branch until Condition True or Count=−1) **命令**† DBcc 命令は反復処理を 1 命令で実現できる．DBcc 命令の実行は反復回数を数えるデータレジスタを用いて行う．分岐条件（5 章，表 5.7 参照）は 16 種類すべてが使用でき，分岐先のアドレス形式は，Bcc 命令と同様にディスプレースメント付/インデックス付プログラム カウンタ相対形式のみである．

DBcc 命令の実行手順を図 15.1 に示す．まず，指定された条件をテストする．その結果が真なら次の命令を実行する．偽ならデータ レジスタから 1 を引く．データ レジスタの値が −1 なら次の命令を実行し，−1 でなければラベルで示す番地へ分岐する．DBcc 命令ではデータ レジスタのサイズをワードとして使用する．

図 15.1　DBcc 命令の実行手順

Bcc 命令が真の場合に分岐するのに対し，DBcc 命令は偽の場合に分岐することに注意しておく必要がある．DBcc 命令は条件が常に偽（F）の場合に，ニモニックが DBRA (Decrement and Branch Always) と書くことができる．すなわち，DBRA 命令は常にデータ レジスタから 1 を引き，−1 なら次の命令に進み，−1 以外なら分岐を実行する命令である．

DBRA 命令と BRA 命令を使用した場合の反復処理について比較する．DBRA 命令を使用すると，

---

† DBcc　Dn, 〈ea〉　cc が真ならば次の命令へ進む．
cc が偽ならば Dn←Dn−1　Dn≠−1 ならば 〈ea〉 へ分岐，Dn=−1 ならば次の命令へ進む．

```
         MOVEQ    #10,D0
LOOP       :
         DBRA     D0,LOOP
```

となり，1命令で反復処理を実現できる．これに対して BRA 命令を使用すると，

```
         MOVEQ    #10,D0
LOOP       :
         SUBQ     #1,D0
         BGE      LOOP
```

となり，SUBQ, BGE 命令の2命令かかる．反復処理ループ内から命令を1つ減じることは実行時間を短縮する上で効果が大きい．

また，DBcc 命令は反復処理のみに使用されるので Bcc 命令よりもプログラムの構造が単純化され，プログラムが見やすくなる．ただし DBcc 命令ではループ カウント用のレジスタを専用に使用する．

15.2 サブルーチン操作命令

68000ではサブルーチン呼出し用の命令として BSR, JSR 命令の2つが用意されており，サブルーチンから戻る命令として RTR, RTS 命令が用意されている．以下，各命令について説明する．

a. JSR (Jump to Subroutine) 命令†1　JSR 命令はプログラム カウンタの内容†2をシステムスタックに格納し，その後指定されたアドレスへ分岐する命令である．JSR 命令の分岐先指定に使用できるアドレス形式は，JMP 命令と同様に，アドレス レジスタ間接形式，ディプレーメント付アドレス レジスタ間接形式，インデックス付アドレス レジスタ間接形式，長および短絶対アドレス形式，ディスプレースメト付/インデックス付プログラムカウンタ相対形式の6種類である．

JSR 命令では，ディスプレースメント付プログラム カウンタ相対形式のときは，2ワード命令となる．

b. BSR (Branch to Subroutine) 命令†3　BSR 命令はプログラム カウンタの内容

---

†1 JSR 〈ea〉　　PC ⟶ SP @
　　　　　　　　デスティネーション ⟶ PC
†2 JSR 命令の次の命令を指している．
†3 BSR 〈ラベル〉　PC ⟶ SP @−
　　　　　　　　PC+d → PC

をシステム スタックに格納し，その後指定したラベルに分岐する命令である．BSR 命令の分岐先指定に使用できるアドレス形式は，Bcc, BRA 命令と同様に，ディスプレースメント付/インデックス付プログラム カウンタ相対形式である．

また，BRA 命令は Bcc, BRA 命令と同様に分岐する範囲により命令長の選択ができる．すなわち，分岐する範囲が $PC-2^7 \sim PC+2^7-1$ の場合には 1 ワード命令，$CP-2^{15} \sim PC+2^{15}-1$ の場合には 2 ワード命令を選択できる．

c. RTS (Return from Subroutine) 命令†1　RTS 命令はサブルーチンから戻るための命令である．RTS 命令を実行することにより，システム スタックから戻り先のプログラム実行番地が読み出され，プログラム カウンタにセットされる．

RTS 命令を用いるときのサブルーチンの動作例を図 15.2 (a) に示す．

一般に，この命令は，JSR, BSR 命令によってシステム スタックに退避されたプログラムカウンタの内容を回復する場合に用いられる．

(a) RTS 命令を用いたサブルーチン　　(b) RTR 命令を用いたサブルーチン

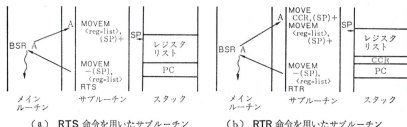



図 15.2 サブルーチン呼出し時の動作例

d. RTR (Return from Subroutine and Restore CC) 命令†2　RTR 命令はシステムスタックからコンディション コード レジスタとプログラム カウンタの内容を回復する命令である．このときステータス レジスタのシステム バイト (上位 8 ビット) は影響を受けない．

RTR 命令を用いたサブルーチンの動作例を図 15.2 (b) に示す．この場合はサブルーチンの先頭で MOVE from SR 命令を用いてコンディション コードレジスタを退避しておくことが必要である．

---

†1 RTS (オペランドなし)　SP@＋──→PC
†2 RTR (オペランドなし)　SP@＋cc
　　　　　　　　　　　　　SP@＋PC

15.3 引数の受渡し方法

本節ではサブルーチンの引数の受渡し方法の代表的なものについて述べる．引数の受渡し方法には値呼び，番地呼び，アドレス リストのポインタを渡す方法などがある．以下にこれらの概要について述べる．

15.3.1 値　呼　び

値呼びはパラメータを実際の値で受渡しするものであり，一般にメインルーチンからサブルーチンの一方通行の方法である．以下に値呼びを実現する方法を示す．

a. データ レジスタを用いる方法　　データ レジスタを用いて値呼びを実現する方法は操作も簡単で，実行時間も短い．すなわち，68000のアドレス形式の中で最も実行時間の短いデータ レジスタ直接アドレス形式を用いることにより，引数の受渡しにかかる時間を短縮できる．データ レジスタを用いた値呼びのプログラム例を以下に示す．

```
MOVE.W   引数1，D0
MOVE.L   引数2，D1
BSR      〈サブルーチン〉
```

このプログラムを実行した場合のレジスタの状態を図15.3に示す．サブルーチンで引数を参照する場合はデータ レジスタ直接アドレス形式を用いる．

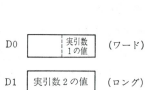



図 15.3　サブルーチンの引数（データ）をレジスタを用いて渡す方法

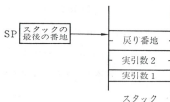



図 15.4　サブルーチンの引数（データ）をスタックを用いて渡す方法

b. スタックを用いる方法　　次にスタックを用いて値呼びを実現する方法を示す．メインルーチンで引数をスタックに格納する場合には，プリデクリメント アドレス レジスタ間接アドレス形式を用いる．引数をスタックに格納した後にサブルーチンへジャンプする．スタックを用いた値引数のプログラム例を示す．

```
MOVE.W   引数1，－(SP)      引数1 → スタック
MOVE.L   引数2，－(SP)      引数2 → スタック
```

```
BSR        〈サブルーチン名〉
```

このプログラムを実行したときスタックの状態を図15.4に示す．スタックにはMOVE命令で引数が格納され，次にBSR命令で戻り番地が格納される．サブルーチンで引数を参照する場合は，スタック ポインタをベース レジスタとして用い，ディスプレースメント付アドレス レジスタ間接アドレス形式を用いる．すなわち，

```
MOVE.W   8(SP),D0     引数1→D0
MOVE.L   4(SP),D1     引数2→D1
```

で引数をデータ レジスタに転送できる．

c．プログラムの一部を用いる方法 プログラム領域中に仮引数の領域をとることにより値呼びを実現する方法を示す．この方法の利点はプログラム領域中に仮引数の領域をとるので，プログラムと仮引数の領域を別々に管理する手間が省けることである．通常仮引数の領域をJSR, BSR命令の直後に置く．メインルーチンではサブルーチン呼び出す前に仮引数の領域に仮引数をセットしておく．たとえば，

```
BSR      〈サブルーチン名〉
DC.W     $10
```

とする．

サブルーチンからメインルーチンへ戻る場合には，図15.5に示すようにJSR, BSR命令の直後に戻るのではなく，仮引数の領域の次の番地へ戻らなければならない．そこで，JSR, BSR命令でスタックに格納した戻り番地（プログラム カウンタの値）に仮引数の領域分だけ加える操作を行わなければならない．以下にプログラム例を示す．

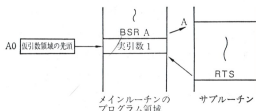



図15.5 サブルーチンの引数(データ)をプログラムの一部を用いて渡す方法

```
MOVEA.L   (SP),A0
ADDQ.L    #2,(SP)
MOVE.W    (A0),D1
```

まず，JSR, BSR命令でスタックに格納した戻り番地はサブルーチン内で引数を参照するために必要なので，MOVEA命令を用いてアドレス レジスタA0に格納する．次に，ス

タックに格納した戻り番地に仮引数の領域のバイト数を加える．引数を参照する場合は，A0を用いアドレス レジスタ間接アドレス形式で参照する．

15.3.2 番 地 呼 び

番地呼びはパラメータをそれが格納されているメモリの番地で受け渡しするものであり，メインルーチンとサブルーチンの間の両方向のパラメータの受渡しが可能な方法である．以下番地呼びを実現する方法を示す．

a. アドレス レジスタを用いる方法　番地呼びでは実引数の番地が仮引数になるのでアドレス レジスタを用いる．アドレス レジスタに実引数の番地をセットするのにLEA命令を用いる．以下にアドレス レジスタを用いた番地呼びの受渡し方法のプログラム例を示す．

```
LEA    引数1，A0
LEA    引数2，A1
BSR    〈サブルーチン名〉
```

このプログラムを実行した場合のレジスタの状態を図15.6に示す．アドレス レジスタ A0，A1にはLEA命令によって格納された実引数1，2の番地が入っている．サブルーチンで引数を参照する場合はアドレス レジスタ間接アドレス形式を用いる．この方法はアドレスレジスタを引数の受渡しに使用できるようにレジンタの割当てを行う必要がある．また，引数の数が多い場合にはよい方法ではない．

A0 実引数1の番地
A1 実引数2の番地
実引数2
実引数1
メインルーチンの変数領域

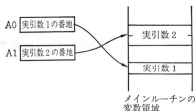



図 15.6 サブルーチンの引数（アドレス）をアドレス レジスタを用いて渡す方法

b. スタックを用いる方法　番地呼びをスタックを用いて実現するにはPEA命令を用いる．PEA命令で実引数の番地をスタックに格納する．スタックを用いる方法のプログラム例を以下に示す．

```
PEA    引数1
PEA    引数2
BSR    〈サブルーチン名〉
```

図15.7にこのプログラムを実行したときのスタックの状態を示す．スタックにはPEA命令によって，実引数1，2の番地が格納される．サブルーチン内でこれらの引数の番地を参照する場合には，スタック ポインタをベース レジスタとして用い，ディスプレース

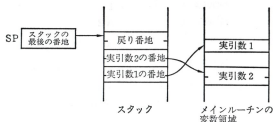



図 15.7 サブルーチンの引数（アドレス）をスタックを用いて渡す方法

メント付アドレス レジスタ間接アドレス形式を用いる．このプログラム例を以下に示す．

```
MOVEA.L    4(SP),A1
MOVEA.L    8(SP),A0
```

MOVEA 命令により，アドレス レジスタ A0, A1 に引数の番地がセットされる．実際に引数を参照するには，さらに，

```
MOVE.W    (A0),D0
MOVE.L    (A1),D1
```

とする．この MOVE 命令でデータ レジスタ D0, D1 に引数の値がセットされる．

15.3.3 アドレス レジスタをポインタとして渡す方法

この方法はテーブルに実引数のアドレスをあらかじめセットしておき，そのテーブルの先頭番地をアドレスレジスタにセットして，引数の受渡しを1個のアドレス レジスタで行うものである．この方法の利点は引数の受渡しにアドレス レジスタ1個で済むことである．アドレス リストをポインタで渡す場合のレジスタ，テーブルの状態を図 15.8 に示す．

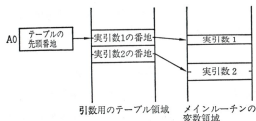



図 15.8 サブルーチンの引数をアドレス リストのポインタを用いて渡す方法

たとえば，テーブルの先頭番地が 1000 番地の場合は，

```
MOVEA.L    #1000,A0
```

でアドレス レジスタ A0 にテーブルの先頭番地をセットできる．

次に引数を参照するプログラムを示す．

```
MOVEA.L    (A0),A1
MOVEA.L    4(A0),A2
```

によって，アドレス レジスタ A0，A1 に引数の番地をセットできる．さらに

```
MOVE.W    (A1),D0
MOVE.L    (A2),D1
```

より，引数をそれぞれデータ レジスタ D0，D1 に転送できる．

15.4 LINK/UNLK 命令

LINK, UNLK 命令はシステム スタック上に，ある大きさの領域を確保したり，開放したりするのに用いる．通常，LINK 命令はサブルーチンの先頭でローカル変数領域をシステム スタックに確保するのに用い，UNLK 命令はサブルーチンから戻る直前に確保したシステムスタックの領域を開放するのに用いる．また，LINK, UNLK 命令を用いることにより，リカーシブ，リエントラントなサブルーチンを構成することができる．以下に LINK, UNLK 命令について示す．

a. LINK (Link and Allocate) 命令† LINK 命令はシステム スタック上に最大 $2^{15}-1$ バイトまでの大きさの領域を確保するのに用いられる．LINK 命令のソース オペランドには，システム スタック内のデータを参照するときに用いるベース レジスタ（アドレス レジスタ）を指定し，デスティネーション オペランドには確保する領域の大きさを指定するディスプレースメントを指定する．

ベース レジスタは UNLK 命令で確保した領域が開放されるまで固定しておき，このベース レジスタを用いたディスプレースメント付アドレス レジスタ間接形式およびインデックス付アドレス レジスタ間接形式によって確保したシステム スタック内のデータを参照する．

ディスプレースメントは 16 ビットの符号付整数（$-2^{15}\sim2^{15}-1$）である．

LINK 命令の実行手順を図 15.9 に示す．まず，オペランドで指定したアドレス レジスタの値をシステム スタックに格納する．このスタック操作により，システム スタック ポインタの値は 4 減少する．更新されたこのシステム スタック ポインタの値が上記アドレス レジスタの値となる．そして，システム スタック ポインタにはディスプレースメントの

---

† LINK　An, #〈エリア長〉　An ⟶ −(SP)
　　　　　　　　　　　　　SP ⟶ An
　　　　　　　　　　　　　SP + d ⟶ SP

(a) LINK 前　(b) LINK 後　(c) UNLK 後

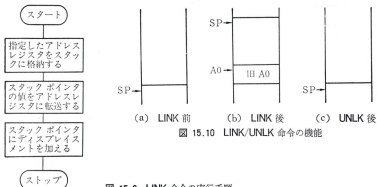



図 15.10 LINK/UNLK 命令の機能

図 15.9 LINK 命令の実行手順

値が加えられる，すなわち，アドレス レジスタが確保された領域の先頭を指し，システム スタック ポインタが最後を指すことになる．

LINK　A0,#−10

という LINK 命令を実行した結果を図 15.10 (a)，(b) に示す．アドレス レジスタ A0 が確保した領域の先頭を指しており，システム スタック ポインタ SP が最後を指している．

b. UNLK (Unlink) 命令†　UNLK 命令は LINK 命令で確保したシステム スタック領域を開放するために用いる．UNLK 命令のデスティネーション オペランドにはアドレス レジスタを指定する．

UNLK 命令の実行手順を示す．オペランドで指定したアドレス レジスタの値(LINK 命令を実行する前のシステム スタック ポインタの値 −4) をシステム スタック ポインタに移す．次に，このシステム スタック ポインタの指すアドレス内容 (LINK 命令のオペランドで指定したアドレス レジスタの LINK 命令を実行する前の値) を UNLK 命令のオペランドで指定したアドレス レジスタに移す．この操作でシステム スタック ポインタの値は 4 増加する．すなわち，UNLK 命令により，LINK 命令で指定したアドレス レジスタとシステム スタック ポインタが LINK 命令を実行する前の状態に戻る．

UNLK　A0

を実行した結果を図 15.10 (c) に示す．システム スタックが LINK 命令実行前と同じ状態に戻っていることがわかる．

次に実際に LINK, UNLK 命令を用いたサブルーチンの構成例を示す．

† UNLK An　An ⟶ SP
(SP)+ ⟶ An

メインルーチン

```
        MOVE.W    D0,-(SP)         引数の受渡し（スタックを用いた値呼び）
        PEA       (A0)             引数の受渡し（スタックを用いた番地呼び）
        JSR       SUB              サブルーチン SUB へ分岐
サブルーチン
SUB     LINK      A6,#-6           サブルーチン内で使用するローカル変数領域
                                   を6バイト確保
        MOVEM     D1/D2,-(SP)      サブルーチン内で使用するデータ レジスタ
                                   D1,D2を退避
          ⋮
        MOVEM     (SP)+,D1/D2      データ レジスタ D1, D2を回復
        UNLK      A6               ローカル変数領域の開放
        RTS                        メインルーチンへ戻る
```

まず，引数の受渡しを示す．この例ではスタックを用いた値呼びと番地呼びを用いている．値呼びの場合は MOVE 命令でシステム スタックに引数の値を格納し，番地呼びの場合は PEA 命令でシステム スタックに引数のアドレスを格納する．次に JSR（または BSR）命令でサブルーチンに分岐する．このときのシステム スタックの状態を図 15.11 (a) に示す．

サブルーチンではサブルーチン内で使用するローカル変数領域をシステム スタック内に確保する．この例では LINK 命令を用いてローカル変数領域を 6 バイト確保している．このときのシステム スタックの状態を図 15.11 (b) に示す．そして，サブルーチン内で使用するレジスタの内容を MOVEM 命令でシステム スタックに退避する．この例ではデー

(a) JSR 実行後　(b) LINK 実行後　(c) MOVEM 実行後　(d) UNLK 実行後

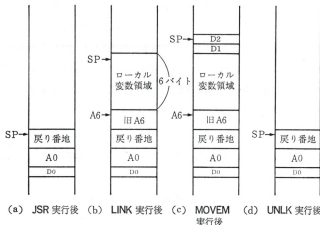



図 15.11　サブルーチン呼出しシーケンス中のシステム スタックの状態

タ レジスタ D1, D2 を退避している. このときのシステム スタック状態を図 15.11 (c) に示す.

サブルーチン内の処理が終了してサブルーチンから戻る場合には, MOVEM 命令で退避しておいたレジスタの値を戻す. 次に, UNLK 命令によって LINK 命令で確保したローカル変数領域を開放する. このときのシステム スタック の状態を図 15.11 (d) に示す.

このように LINK, UNLK 命令を用いることにより, サブルーチン内で用いるローカル変数領域を動的に確保したり, 開放したりすることが可能となる.

15.4 条件セット命令

Scc (Set Conditionally) **命令**† Scc 命令は演算の結果をプログラムの別の場所で知りたい場合に使用する命令である. すなわち, Scc 命令はテスト コンディションを行い, その結果をメモリ内の任意の番地に書き込む命令である. Bcc 命令では, 条件を判定してすぐに次の処理を選択する場合に有効であるが, Scc 命令はその処理を後で行ったり, 結果を記憶しておく場合に有効である.

Scc 命令は "cc" で表す部分にニモニックで条件を指定し, その指定した条件が真 (T) なら実効アドレスで示されたバイトにオール 1 (11111111) をセットし, 条件が偽 (F) ならそのバイトをオール 0 (00000000) にセットする. 条件は 16 種類 (5 章, 表 5.7 参照) 使用できる. たとえば, "等しい" かどうかという条件をチェックする場合には, SEQ というニモニックで命令を書く.

Scc 命令で使用できるアドレス形式はデータ レジスタ直接形式, アドレス レジスタ間接形式, ポスト インクリメント アドレス レジスタ間接形式, プリデクリメント アドレス レジスタ間接形式, ディスプレースメント付/インデックス付アドレス レジスタ間接形式, 長または短絶対アドレス形式である. また, 使用できるオペランド サイズはバイトだけである.

Scc 命令を使用したプログラム例を以下に示す.

```
CMP   #4,D1
SLE   1000
```

このプログラムは D1 の値を 4 と比較し, D1 の値が 4 より小さい場合に 1000 番地をオール 1 にセットするプログラムである.

---

† Scc <ea>　cc が真のとき　オール 1 ⟶ <ea>,　cc が偽のとき　オール 0 ⟶ <ea>

16　システム制御命令

この章ではシステム制御命令として，トラップを発生させる命令，ユーザ状態において ステータス レジスタ/コンディション コード レジスタを操作する命令について説明する．特権命令は17章にて説明する．このほか，NOP 命令および未実装命令についても述べる．

16.1　トラップ発生命令

トラップ発生命令はプログラムからトラップを発生するための命令である．トラップが発生するとプロセッサは例外処理を行い，スーパバイザ状態でトラップに対する処理プログラムを実行する（8章参照）．トラップ発生命令は次のような場合に使用される．

（1）　ユーザ プログラムが入出力動作を OS に依頼するとき

（2）　オーバフロー，0除算などの処理プログラムを起動するとき

（3）　ユーザ プログラムを終了し OS へ制御を移すとき

68000には3種類のトラップ発生命令，すなわち TRAP，TRAPV，CHK 命令が用意されている．以下，各命令ごとに説明を行う．

a． TRAP（Trap）命令† 　TRAP 命令は無条件にトラップを発生させる命令であり，プロセッサは例外処理を行う．TRAP 命令のオペランドにはトラップ ベクタ番号を記述する．トラップ ベクタ番号には0から15までの16とおりを用いることがきる．これらの番号は小さい方から順に32～47番の例外ベクタに対応する．たとえば，次の命令の実行例を図16.1に示す．

```
TRAP    #1
```

プロセッサはプログラム カウンタおよびステータス レジスタをスーパバイザ システム

---

† 　TRAP　#〈トラップ ベクタ番号〉　無条件にトラップを発生

(a) 例外処理　　(b) 例外処理後のトラップルーチンの起動

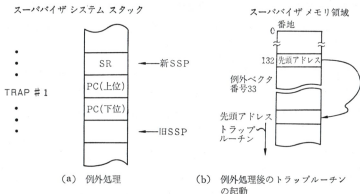



図 16.1 TRAP 命令の実行例

スタックに格納したのち，33番の例外ベクタ，すなわち132番地の内容で示される番地からトラップ処理を開始する（8章参照）．

トラップ処理ルーチンへパラメータを渡すためには，次のプログラム例のように TRAP 命令の直後にパラメータを設定する（15章参照）．この例では50がパラメータである．

```
TRAP    #10
DC.W    50
```

パラメータの受渡し方法は基本的にサブルーチン操作（15章参照）と同じである．

b. TRAPV (Trap if Overflow Set) 命令†1　TRAPV 命令は演算のオーバフローをチェックするための命令であり，V フラグが1のときトラップを発生する．この命令にはオペランドはなく，例外ベクタ番号として7番（28番地に対応）が割り当てられている．オーバフローの処理は28番地の内容で示される番地から実行される．オーバフローが生じていなかった場合には次の命令の実行に移る．

c. CHK (Check Register against Bounds) 命令†2　CHK はメモリアクセスの境界チェックを行うための命令である．境界の侵犯があった場合トラップを発生する．命令のフォーマットを次に示す．

```
CHK    <ea>, Dn
```

---

†1 TRAPV（オペランドなし）		V フラグ=1 のときトラップ発生
†2 CHK 〈ea〉, Dn	Dn<0	N=1，トラップ発生
	0≦Dn≦(〈ea〉)	N=未定義，次の命令へ進出
	Dn>(〈ea〉)	N=0 トラップ発生

ソース オペランドにはアドレス レジスタ直接形式以外のアドレス形式が使用できるが，デスティネーション オペランドはデータ レジスタを指定しなければならない．オペランド サイズはワードのみである．

この命令の実行は次のように行われる．プロセッサはデスティネーション オペランドで指定されたデータ レジスタ Dn を，0ならびにソース オペランドで指定された〈ea〉の内容と比較する．この比較結果により，命令の動作は次の3つの場合に分けることができる．

（1） Dn<0　　　　　　のとき，Nフラグに1を設定しトラップを発生

（2） 0≦Dn≦(〈ea〉)　　のとき，Nフラグは未定義，トラップは発生しない

（3） Dn>(〈ea〉)　　　　のとき，Nフラグに0を設定しトラップを発生

トラップが発生した場合，プロセッサは例外処理を行い，例外ベクタの6番が内部的に生成される．この後，24番地で示される番地から CHK トラップ処理ルーチンが実行される．上記 (1), (3) のどちらの原因でトラップが発生したかは N フラグで判断できる．上記 (2) の場合，プロセッサは CHK 命令の次の命令を実行する．Z, V, C フラグは(1), (2), (3) いずれの場合も未定義である．

CHK 命令の使用例を次に示す．配列 AR は $300 個のバイトデータの配列とし，アドレス レジスタ A1 は AR の先頭番地を示しているとする．AR 内の i 番目 (0≦i≦$2FF) のデータ AR(i) へアクセスするためには，i をデータ レジスタ Dn にセットし，0(A1, Dn) というインデックス付アドレス レジスタ間接形式を使用すればよい．このとき，データ レジスタ Dn は，0≦Dn≦$2FF でなければならない．これをチェックするために

```
CHK    #$2FF,Dn
```

という命令が使用でき，配列外参照のチェックに役立つ．

16.2 コンディション コード操作命令

ユーザ状態でコンディション コードにアクセスする命令として68000には次の命令がある．

（1） CCR と実効アドレス (EA) 間の転送——MOVE EA to CCR, MOVE SR to EA

（2） CCR に対する論理演算——ANDI to CCR, EORI to CCR, ORI to CCR

以下，各命令について説明する．

a. MOVE EA to CCR (Move EA to Codition Codes Register) 命令[†1]　　この

命令は，実効アドレス EA の内容をコンディション コード レジスタへ転送する命令である．ソースオペランドのサイズはワードであるが，下位8ビットのみが CCR へ転送される．実効アドレス EA のアドレス形式はアドレス レジスタ直接形式が禁止されていることを除けば MOVE 命令と同じである．この命令は，CCR の各フラグに値を設定するときに使う．たとえば，10進演算を行う前に，Z フラグを1に設定し，他のフラグをクリアしたい場合

```
MOVE    #4,CCR
```

とすればよい．

b. MOVE SR to EA (Move from Status Register to EA) **命令**†2　この命令はステータス レジスタの内容を実効アドレス EA へ転送する命令である．オペランド サイズはワードのみが許されている．実効アドレス EA のアドレス形式は通常の MOVE 命令と同じである．この命令は CCR をメモリ上に記憶するために使用する．たとえば，CCR をシステム スタック内へ格納する場合

```
MOVE    SR,-(SP)
```

という命令を実行すればよい．この命令例はサブルーチンの先頭で使用される．サブルーチンから戻るときに RTR 命令を用いれば，コンディション コードレジスタもサブルーチン呼出し前の状態に復帰できる．

c. ANDI to CCR (AND Immediate to Codition Codes) **命令**†3　この命令は，コンディション コード レジスタの内容とイミディエイト値の論理積を計算し，コンディション コード レジスタへ格納する命令である．オペランド サイズはバイトのみである．たとえば，X フラグのみをクリアし，他のフラグは元の値のままにするためには，次の命令を実行すればよい．

```
ANDI    #15,CCR
```

また，本命令の実行時間は8サイクルであり，MOVE EA to CCR 命令の実行時間は(12+EA 計算) サイクルである (7章参照)．したがって，特定のフラグをクリアするためには ANDI to CCR 命令の方が MOVE EA to CCR 命令よりも4クロック +EA 計算の分だけ短い．

---

†1 MOVE <ea>, CCR　(<ea>) ⟶ CCR
†2 MOVE SR, <ea>　SR ⟶ <ea>
†3 ANDI #<データ>, CCR　CCR・#<データ> ⟶ CCR

d. EORI to CCR (Exclusive OR Immediate to Condition Code Register) 命令†1

この命令はコンディション コード レジスタ の内容とイミディエイト値の排他的論理和を計算しコンディション コード レジスタへ格納する命令である．オペランド サイズはバイトのみである．たとえば，X フラグのみを反転させたいときは次の命令を実行すればよい．

```
EORI    #16,CCR
```

e. ORI to CCR (OR Immediate to Condition Code Register) 命令†2　この命令はコンディション コード レジスタの内容とイミディエイト値の論理和を計算し，コンディションコード レジスタへ格納する命令である．オペランドサイズはバイトのみである．たとえば他のフラグに影響を与えず X フラグに 1 を設定したいときは次の命令を実行すればよい．

```
ORI    #16,CCR
```

16.3 その他の命令

本節では，前節までに説明できなかった命令のうち，NOP 命令ならびに未実装命令について説明する．

a. NOP (No Operation) 命令†3　NOP 命令は何もしない命令である．命令長は 1 ワードであり実行時間は 4 クロック サイクルである．この命令は主として，プログラム中の領域確保ならびに，時間遅れとして利用される．

b. 未実装命令　未実装命令はオペレーション ワードの上位 4 ビットが "1010" または "1111" のビットパターンをもつ命令である（8 章参照）．8 章でも説明したとおり，未実装命令は例外処理を引き起す．そのため未実装命令処理ルーチンに何らかの処理を行うプログラムを割り当てておけば，未実装命令が未実装処理ルーチン中のある種の処理を行うようにみえる．すなわち，いわゆるエミュレーションを行わせることが可能となる．

例として浮動小数点演算エミュレーション命令のフォーマットを図 16.2 に示す．図 16.2 の命令はデータ レジスタ D0 と D4 の浮動小数点演算を行うものであり，図中オペレーション フィールドで演算の種類を示す．第 0～2 および第 5～8 ビットの内容は 0 ま

---

†1 EORI　#<データ>, CCR　　CCR ⊕ #<データ> ⟶ CCR
†2 ORI　#<データ>, CCR　　CCR ∨ #<データ> ⟶ CCR
†3 NOP　(オペランドなし)　　何もしない

たは1のいずれであってもよい．オペレーション フィールドのビット パターンと演算の指定を次のように対応づける．

ビット4	ビット3		
0	0	FADD	浮動小数点加算
0	1	FSUB	浮動小数点減算
1	0	FMUL	浮動小数点乗算
1	1	FDIV	浮動小数点除算

データレジスタ D0 と D4 の浮動小数点加算命令をアセンブラ レベルで記述すると

```
FADD    D0,D4
```

となる．この場合，アセンブラはこの命令を図 16.2 の命令フォーマットに変換できるものでなければならない．

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	×	×	×	×	×	×	×			×	×	×

オペレーション フィールド（ビット4, 3）

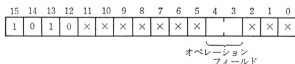



図 16.2　浮動小数点演算のエミュレーション用命令フォーマット例

以下に，浮動小数点演算エミュレーション用の初期化ルーチンの例を示す．命令の上位4ビットが "1010" の命令は10番の例外ベクタを発生する．このため10番の例外ベクタに対応する40番地には初期化ルーチンの先頭アドレス1500番地を格納しておく．実際の初期化ルーチンは1500番地から始まる．この例では演算の種類を調べそれぞれの演算プログラムへジャンプさせるものである．なお，データ レジスタ D7 およびアドレス レジスタ A5 の値は破壊される．

```
        ORG         40              1010命令の例外ベクトル番号
        DC.L        1500            例外処理ルーチンの開始番地
*
        ORG         1500
        MOVEA.L     2(SP),A5        スタックから1010命令のプログ
                                    ラムカウンタ A5 へ転送
        MOVE.W      (A5),D7         1010命令の命令コードをD7へ転
                                    送
        MOVE.W      D7,-(SP)        1010命令の命令コードをスタッ
                                    クへ退避
        ANDI        #$18,D7         演算フィールド抽出
        LSR         #1,D7           演算フィールドを0, 4, 8, 12の
```

```
                                            いずれかに変換
            LEA        OPANADDR,A5          オペレーションの先頭番地をA5
                                            へロード
            MOVEA.L    0(A5,D7.W),A5        A5の番地を演算フィールドで修
                                            飾し，ジャンプ先をA5へロード
            JMP        (A5)                 各演算処理ルーチンへジャンプ
*
OPANADDR    DC.L       FADD                 加算処理ルーチン
            DC.L       FSUB                 減算処理ルーチン
            DC.L       FMUL                 乗算処理ルーチン
            DC.L       FDIV                 除算処理ルーチン
```

17　特　権　命　令

8章では，68000に2種類のプログラム実行状態，すなわち，ユーザ状態とスーパバイザ状態とがあることを説明した．本章では，スーパバイザ状態（ステータス レジスタのSビットが“1”）でのみ使用可能な特権命令とよばれる命令について説明する．

68000でプログラムを実行状態を2種類に区別したのは，一般のプログラムの実行とシステムを管理するプログラム（一般にはOSと称されるシステム プログラム）の実行とを区別し，システム全体の信頼性を向上させるためである．この手法は，すでに大形計算機では一般的となっている．特権命令は，システムを管理するための命令であるので，特にこの命令の実行は他の命令の実行と区別する必要がある．特権命令を大きく分類すると，次のようになる．

（1）ステータス レジスタの内容を変更（ANDI, EORI, ORI, MOVE EA to SR 命令）

（2）プログラム実行状態の変化に伴う相互間連絡（MOVE to/from USP 命令）

（3）タスクの切換えおよびその準備等（RTE, RESET, STOP 命令）

特権命令はスーパバイザ状態でのみ使用可能であり，ユーザ状態で使用しようとした場合には，命令実行の段階で特権例外が発生する．特権例外によって実行される特権違反例外処理に関しては8.3節（106ページ）を参照されたい．

a. ANDI to SR (AND Immediate to Status Register)[†1], EORI to SR (Exclusive OR Immediate to Status Register)[†2], ORI to SR (Inclusive OR Im mediate to Status Register)[†3]**命令**　これらの命令は，ステータス レジスタの内容（S・Tビット，割込みマスク情報，コンディション コード）とイミディエイトデータとの論理演算を

---

†1 ANDI　#〈データ〉, SR　　SR・#〈データ〉⟶SR
†2 EORI　#〈データ〉, SR　　SR⊕#〈データ〉⟶SR
†3 ORI　#〈データ〉, SR　　SR+#〈データ〉⟶SR

行い，その結果に従ってステータス レジスタの内容を変更する命令である．この命令の実行により，プログラム実行状態の変更，トレース処理の実行，割込みレベルの設定，コンディション コードの初期設定が可能となる．本命令が，一般命令である“to CCR”と違う点は，“to CCR”の場合には無視されたイミディエイト データ上位バイト（ビット番号 15～8）が，本命令ではシステム バイトの情報を変更させる点である．システム バイトの情報は，プログラム実行状態の変更，トレース処理の実行，割込みレベルの設定を行うため，システムに直接影響を与える．

システム バイトの情報変更によるシステム コントロールの一例として，外部割込みによるタスク切換えの制御について簡単にふれる．たとえば，現在実行中のシステム プログラムおよびそれに続いてソフト的に起動されるプログラムにおいて，割込みレベル 7 以外の外部割込みは許したくないとする．この場合，システム バイトの割込みマスク情報だけを次の命令によって変更させ，レベル 7 とする．

```
ORI    #$0700,SR
```

また逆に，すべての外部割込みに対して応答させようと考えた場合には，次の命令を実行して割込みマスク情報をレベルゼロに変更する．

```
ANDI    #$F0FF,SR
```

ファンクションコード FC 0～2 の出力をアドレスの一部として直接用いる場合には，ステータス レジスタ S ビットの情報を変える必要が生ずる．すなわち，S ビットの情報が FC 2 の出力に直接反映されるため，スーパバイザ状態で実行されるシステム プログラムがユーザ状態で実行されたプログラムによって生成されたメモリ上のデータを，参照できないという不都合が起ることがある．そこで，どうしてもやむを得ない場合この命令を用いて S ビット情報を“0”に変更し，システム プログラムのプログラム実行状態をユーザ状態としてから，参照すべきデータを 68000 の内部レジスタ（D 0～7）等に格納し，再度 TRAP 命令の実行によって実行状態をスーパバイザ状態に遷移させる手続きが必要となる．しかし，68000 上位 16 ビット マイクロコンピュータ 68010 では，スーパバイザ状態においては FC 0～2 の出力をプログラムで設定可能となり，この手続きも不要となる．

b.　MOVE EA to SR (Move EA to Status Register) 命令[†]　本命令は，ステータス レジスタの内容を任意の値にセットする命令である．a. で説明した命令は，命令実行前のステータス レジスタの内容と論理演算するので，セットされる値は以前の状態を反映する．それに対し，この命令では，ステータス レジスタの内容を無条件に指定された値

† MOVE 〈ea〉, SR　　〈ea〉⟶ SR

(ソース オペランド) とする.

たとえば，ステータス レジスタにオール0をセットしようとした場合を考える．ステータス レジスタをオール0にすると，プログラム実行状態はユーザ状態に，トレースは不実行の状態に，割込みマスクはレベル0に，そしてコンディション コードはすべて“0”の状態となる．これは，次の命令を実行すればよい．

```
MOVE    #$0000,SR
```

本命令では，ソースオペランドのアドレス形式として，アドレス レジスタ直接形式以外のすべての形式をとることができる．

c. MOVE USP (Move User Stack Pointer) 命令[†1]　本命令は，ユーザ状態とスーパバイザ状態との間でシステム スタック ポインタの値を連絡するためにある．68000ではシステム スタック ポインタとしてアドレス レジスタ A7を利用している．システム スタック ポインタはユーザ状態では USP，スーパバイザ状態では SSP と区別されている．このように，アドレス レジスタ A7は，実際には2つのレジスタで構成されており，どちらのレジスタを使用するかはプログラム実行状態 (ステータス レジスタ S ビットの値) で自動的に決められてしまう．スーパバイザ状態で実行するシステム プログラムにおいては，すべてこのデータを参照可能とする必要がある．そのため，スーパバイザ状態で実行するプログラムで，ユーザ状態で使用したアドレス レジスタ A7すなわち USP をアクセスする手段が必要となる．この手段として利用されるのがこの命令である．

アドレス レジスタ A0〜6は，スーパバイザ状態でも，ユーザ状態でも同じものである．したがってスーパバイザ状態でのユーザ状態における アドレス レジスタ A7 (USP) の参照は，これらアドレス レジスタ A0〜6を仲介として用いる．本命令は次に示すアドレス形式のみが許されている．

```
MOVE.L    An,USP    An → USP
MOVE.L    USP,An    USP → An
```

d. RTE (Return from Exception) 命令[†2]　本命令は，例外処理後に実行されるシステムプログラムから，例外処理を発生させたプログラムに制御を移すためのものである．8章の例外処理で説明したように，例外処理中に68000内のプログラム カウンタおよびステータス レジスタの内容はスーパバイザ システム スタック上 (SSP が指定するメ

---

†1 MOVE An,USP　An ⟶ USP
　MOVE USP,An　USP ⟶ An
†2 RTE (オペランドなし)　(SP)+ ⟶ SR
　(SP)+ ⟶ PC，ただし SP は SSP

モリ上）に退避される．この退避された値を68000内に再び戻すと，例外処理を発生させた命令の次の命令を実行することができる．そのため，例外処理後に実行するシステム プログラムの最後で本命令を実行すれば，例外処理を発生させたプログラムの再開が可能である．

なお，アドレス エラー，バス エラーの例外処理においてシステム スタック上に退避されるデータは，プログラム カウンタおよびステータス レジスタの内容だけではないので，スタック ポインタの内容はプログラム カウンタおよびステータス レジスタの格納アドレスを示していない．また，例外処理後のシステム プログラムでアドレス レジスタA7およびシステム スタック ポインタ（指定法は異なるが同一のレジスタ）を使用した場合にも同様の状況が考えられる．したがって，本命令を実行する前にはスタック ポインタの内容をチェックしておくことが必要である．

e．RESET（Reset External Devices）命令†1　本命令を実行すると68000のリセット ピン出力がアサートされ，このピンに接続された外部デバイスをリセットすることができる．リセットピン出力は124クロック分の期間アサートされている．外部デバイスのリセットは，本命令を実行し，124クロック分の期間に完了するようにする．

本命令の実行によって68000の内部レジスタが影響を受けることはない．また，次サイクル以降の命令読出しは，本命令の実行サイクルの一番最後で行われるため，リセット信号の供給が命令読出しに起因したバス エラーおよびアドレスエラーによって中断されることもない．本命令の実行では，リセット例外処理が行われない．

f．STOP（Load SR Stop）命令†2　STOP命令は，必要な割込みが発生するまでプロセッサをストップさせ，タスク間で同期をとるため等に使用される．

命令の実行では，まず，オペレーション ワードに続く2バイトのイミディエイト データをそのままステータス レジスタにセットする．これにより，ステータス レジスタ内の割込みマスクビットI0～2（ビット番号8～10）もある値に設定される．その後，プロセッサは，この割込みマスクビットに設定したレベルより高レベルの外部割込み要求が行われるまで待機する（ストップ状態）．下記の命令を実行して，割込みマスク情報をレベル7にセットした場合には，レベル7の外部割込み要求および外部リセット信号の入力がないかぎり，プロセッサは再動作を行わない．

---

†1　RESET　（オペランドなし）　　RES端子をアサート
†2　STOP　#〈データ〉　　#〈データ〉⟶SR，割込みまでストップ状態

STOP　#$7F00　T="0", S="1", I0〜I2="1", CC="0" をSRにセット，停止状態となる

本命令によってタスク間で同期をとる場合としては次のような状況が考えられる．すなわち，68000が所定の演算を終了し，I/O待ちとなる場合や，複数の68000を用いてマルチプロセッサを構成し，処理の順序性が問題となるプログラムを実行する場合等である．このような状況の場合には，本命令を用い，割込み方式によって同期をとることが有効である．一方，このような機能をプログラムだけで実現しようとしたのがTAS命令である．TAS命令に関する詳細は13章を参照されたい．

18 例外処理後のシステム プログラム

8章で述べたように，68000では例外処理をプロセッサ内のマイクロプログラムにより実行する．しかし，その処理後にシステム運用上で必要となる種々の機能を果すシステムプログラム（OS）は，それぞれのシステム構成に応じてユーザが用意しなければならない．このシステム プログラムの概要について，この章で簡単に説明する．

例外処理後に実行されるシステムプログラムのアドレスは，それぞれの例外処理発生要因別の例外ベクタによって指定される．例外処理では例外ベクタを読み出し，それぞれのシステム プログラムの実行準備までを行う（8章参照）．つまり，例外処理後に実行するシステム プログラムの起動を，例外処理によって行うのである．一方，例外処理後のシステム プログラムの実行によって，例外処理を発生させたエラー要因の回復を行う．さらにその後のタスクの変更（例外処理を発生させたプログラムの再起動あるいは全く別のプログラムの実行）もシステム プログラムが行うことになる．これは，RTE命令，RTR命令，RTS命令等を用いて実行する．

各例外処理後に実行されるべきシステム プログラムの基本機能，動作について以下説明していく．

18.1 リセット例外処理後のシステム プログラム

本システム プログラムの主要機能は大別すると，次の2点である．

（1） システム全体の初期状態設定

（2） 次に実行すべきプログラム（タスク）の選択

上記（1）の機能としては，イニシャル プログラム ローディング（IPL）等があり，これらはシステム構成によって個々に準備する必要がある．一方，（2）の機能を有するシス

テム プログラムは一般にディスパッチャ (Dispatcher) とよばれている.

ディスパッチャは実行可能なプログラムのリストを調べ，その中で一番優先度が高いプログラムの実行を準備する．プログラム実行の準備として，システム スタック上に，そのプログラムの先頭アドレスおよび初期値とすべきステータス レジスタの内容等をセットする．その後，RTE 命令等を実行して，次に実行すべきプログラムに起動をかける.

実行可能なプログラムがない場合には，プロセッサを待機させる必要がある．実行可能なプログラムの発生を外部割込みの形でプロセッサに知らせる場合には，プロセッサに STOP 命令を実行させて割込み発生まで停止させればよい．また，実行可能なプログラムの発生をメモリ上のある特定なビットの変化によってプロセッサに知らせる場合には，無限ループ内で TAS 命令を実行して，そのビットをチェックし，その結果に従ってループを脱出するようにプログラムを構成すればよい．この無限ループ内のプログラムの実行が待機状態と同じ働きをする.

18.2 割込み例外処理後のシステム プログラム

本システム プログラムの主要機能は，外部デバイスとの交信である．たとえば，I/O 装置が入出力を終えた場合には，その終了を割込みによってプロセッサに知らせる．これら外部デバイスと 68000 との交信方式は，それぞれの外部デバイスに依存しているため一様ではない．そのため，それぞれの外部デバイスに適したシステム プログラムを起動する必要がある.

割込み例外処理によって起動されるシステム プログラムは，8.3 節で説明したように，割込みベクタを用いることにより複数個設定することができる．したがって，タイプの異なる外部デバイスごとに，それぞれに対応したシステム プログラムを準備することができる．そして，それらの起動は，ハード的（あるいはソフト的）に割込みベクタを指定（割込み例外処理中の割込みアクノレッジ サイクル）することで可能となる.

なお，割込み例外処理においてオートベクタを指定（割込みアクノレッジ サイクルで $\overline{\text{VPA}}$ をアサート）した場合には，ベクタ番号 25〜31 が自動的に 68000 内で生成される．また，割込みベクタの読出しを失敗した場合には，スプリアス割込みにより，ベクタ番号 24 が自動的に生成される．これらの事項は，8.3 節で説明した.

18.3 トラップ例外処理後のシステム プログラム

本システム プログラムの主要機能は，ユーザ状態で実行されるプログラムからのI/O要求，演算結果がオーバフローした場合の対策，メモリ間の管理等と種々雑多である．一般に，トラップ発生命令（TRAP, TRAPV, CHK 命令）は，ユーザ状態で実行されるプログラムがスーパバイザ状態で実行されるシステム プログラムを呼び出す（スーパバイザコールと称す場合が多い）場合に用いられることが多い．そのため，その機能はさまざまなものとなる．この種々雑多なシステム プログラムをそれぞれの目的に合せて呼び出せるよう，68000では，トラップ例外処理内で次に実行するシステム プログラムのアドレスを指定するために読み出す例外ベクタを19種（TRAP 命令で16種，TRAPV, CHK 命令，除数0で各1種）設定できるようにしている．

18.4 不当命令および未実装命令例外処理後のシステム プログラム

不当命令例外処理後のシステム プログラムの機能としては，次にあげる2種類等が考えられる．このうち，どの機能を選択するかはシステム設計者の判断で決る．

（1） 不当命令があったことをメモリ上に記録し，その命令の実行は行わず，次の命令から実行を再開する．

（2） 不当命令があったことをメモリ上に記録し，TRAP 命令によってディスパッチを起動させ，別のプログラム（タスク）の実行を行う．

一方，未実装命令例外処理後に起動されるシステム プログラムは，エミュレーション用のプログラムである．このことは，8.3節，16.3節でも説明した．この場合，このシステムプログラムではたとえば浮動小数点演算等をサポートし，最後に，RET 命令で例外処理を発生させたプログラムに戻ることになる．

18.5 特権違反例外処理後のシステム プログラム

本システム プログラムの主要機能は，前節で説明した不当命令例外処理後のシステム プログラムとほぼ同様である．特権違反例外のあったことをメモリ上に記録し，その命令の実行を行わず次の命令の実行から再開，あるいは別のプログラム（タスク）を起動させる

ことがふつうである.

18.6 トレース例外処理後のシステム プログラム

本システム プログラムの主要機能は，1命令実行ごとに68000内の情報をメモリ上に記録するトレース機能である．このトレース機能がプログラム開発の段階では大きな助けとなる（8.3節参照）．トレース例外処理では，トレース機能を有する本システム プログラムを起動させる．

68000内ユーザ レジスタの情報をメモリ上に格納するのにはMOVEM命令が有効である．詳細に関しては12章を参照されたい．また，プログラム カウンタおよびステータス レジスタの内容はスーパバイザ システム スタック（SSPがアドレスを指定するメモリ空間）上に格納されているので，これらの情報も併せて転送しておくと，プログラムのデバックにとって都合がよい．

18.7 バス エラー例外処理後のシステム プログラム

本システム プログラムの主要機能は，バス エラーからの回復である．バス エラー例外処理では，バス エラーを発生させた時点でのプログラム カウンタ，ステータス レジスタの内容の他に，そのとき実行していた命令のオペレーション ワード，参照アドレスおよびそのアクセス タイプの情報がシステム スタック上に退避される（8.3節参照）．したがって，これらの情報を有効に利用してエラー回復を図ればよいのであるが，現状68000では，プログラム カウンタによって示されるアドレスが，バス エラーを発生させた命令の拡張部から次に実行を予定している命令の拡張部までのどの部分を指定しているのか定義できない．そのため，一番簡単なエラー回復の手段はシステム全体をリセットするという方式である．68000上位16ビットマイクロコンピュータ68010のバス エラー例外処理の仕様は，もっとエラー回復に都合の良い仕様，すなわち，バス エラーを発生させた命令を再実行しやすい仕様に改善されることが予想される．

18.8 アドレス エラー例外処理後のシステム プログラム

本システム プログラムの主要機能は，アドレス エラーからの回復である．しかし，ア

ドレス エラー例外処理もバスエラー例外処理と同様，退避されるプログラム カウンタの内容はエラー発生時のアドレスを限定できない．そのため，システム全体をリセットするという方式をとらずにエラー回復を図ろうとした場合には，オペレーション ワード，参照アドレスおよびそのアクセス タイプの情報を有効に利用する必要がある．

この章でのシステム プログラムの説明は概説的なものであり，これらプログラムの仕様はシステム構成，目的によって様々に変化する．そのため，この章で説明をしなかった機能がシステム プログラムに導入される場合もあれば，逆に，ここで説明した機能が省略される場合もある．現在，68000を用いたボード コンピュータ H680SBC システムでは，前述のようなシステム プログラム（リアルタイムモニタ RMS，S680RMS1R）をEPROMの形で提供している（Ⅲ編参照）．

19　プログラム例

前章までは主として各命令毎に機能の説明を行ったが，本章では各命令の理解を深めるために，いろいろな種類の命令を組み合せた，より実際的なプログラム例を示す．プログラム例として，2進化10進数の真数変換，1の個数の数え上げ，文字列の一致判定，ASCIIから2進数への変換，パリティの生成をあげる．これらはいずれも基本的なものばかりであるが，前章までの補足という意味も兼ね備えている．各命令の説明を適宜参照することにより，理解が一層深まるであろう．

19.1　2進化10進数の真数変換

正の2進化10進数の減算において，被減数よりも減数のほうが大きいと演算結果は負になる．ところが，SBCD命令を用いてこの減算を行うと演算結果は10の補数表現になる．たとえば13章，図13.10で示したように12340000−12345678の減算ではボローが発生し，その結果は99994322である．本節では補数表現された演算結果を真数に直すプログラム例を示す．

補数を真数に直すには図19.1に示すようにゼロから補数を引けばよい．そこで，NBCD

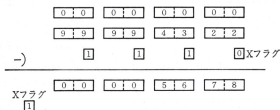



図 19.1　2進化10進化真数変換の数値例

命令を用いた次のルーチンを用いれば結果を真数に変換することができる．なお本来なら符号表示バイトを定義して，これを処理すべきであるがここでは省略した．なおレジスタA1はSBCD命令の例と同様に2進化10進数データの格納されたメモリを指し示すレジスタである．

```
          BCC       NEXT        減算結果でボローがなければ次へ
          ADDQ.L    #4,A1       下位桁を指し示す
          MOVEQ.L   #3,D1       ループ カウンタをセット
          ANDI      #$0F,CCR    X フラグをクリア
L01:      NBCD      -(A1)       10進補数
          DBRA      D0,L01      ループ
NEXT:     EQU       *           補正終了
```

19.2　1の個数の数え上げ

本節では，データ レジスタD0中に含まれている1の個数を数え上げる方法について示す．D0は32ビット長として使用する．したがってD0に含まれる1の個数は0個から最大32個までである．D0中の1の個数を数え上げるためには，論理シフトあるいはローテート命令を使う方法（14章参照）もあるが，本節では別の方法を示す．

図19.2はここで用いたアルゴリズムについて説明した図である．データ レジスタD2にはD0から算術的に1を減算した内容が入っている．D0では第0ビットから第3ビットが0で第4ビットが1であるのに対し，D2では第0ビットから第3ビットまでが1で第4ビットが0であることに注目されたい．したがって，D0とD2の論理積D0･D2においては，第0ビットから第4ビットまですべて0となる．これをもとのD0と比較すると，D0に含まれる1のうち最も右にあるものが1個だけ0に反転していることがわかる．したがって，D0とD0から1だけ減算したD2の論理積を新たなD0とする操作を，D0が0になるまで繰り返せば1の個数を数え上げることができる．D0に含まれている1の個数が少ない場合には，このアルゴリズムは実行効率がよい．プログラム例を次に示

図 19.2　データ レジスタD0に含まれる1の個数を数え上げる方法

す．この例ではデータ レジスタ D1 に 1 の個数が入るものとする．

```
      CLR.L    D1        D1 をクリア
      TST.L    D0        D0 が 0 か？
      BEQ      QUIT
L1:   ADDQ.L   #1,D1     D1 に 1 を加える
      MOVE.L   D0,D2     D0 へ D2 へコピーする
      SUBQ.L   #1,D2     D2 から 1 を減算する
      AND.L    D2,D0     D0 と D2 の論理積をとる
      BNE      L1
QUIT: NOP
```

19.3 文字列の一致判定

本節では，長さ 80 文字（80 バイト）の STRING 1 および STRING 2 の一致判定を行うプログラム例を示す．各文字列はメモリ上にあるものとする．一致判定に際しては，CMPM 命令と DBNE 命令を用いる．プログラム例を次に示す．

```
          LEA      STRING1,A0        A0 に文字列 1 の先頭番地をセットする
          LEA      STRING2,A1        A1 に文字列 2 の先頭番地をセットする
          MOVEI.L  #19.D0            D0 に文字列の長さ −1 をセットする
LOOP:     CMPM.L   (A0)+,(A1)+       文字列 1 と文字列 2 を 1 文字ずつ比較する
          DBNE     D0,LOOP           一致している間は分岐し比較を続ける
          BMI      D0,SAME           D0 はマイナスの値か？
            ⋮
STRING1:  DC       'COMPUTER…'       文字列 1 （80 バイト）
STRING2:  DC       'OVERFLOW…'       文字列 2 （80 バイト）
```

まず，LEA 命令を用いて 2 つの文字列の先頭番地をそれぞれアドレス レジスタ A0, A1 にセットする．次にデータ レジスタ D0 に比較する文字列の長さ −1（ここでは #19）をセットする．文字列の比較には CMPM 命令を用いる．CMPM 命令のアドレス形式はポスト インクリメント アドレス レジスタ間接形式のみであり，命令実行毎にアドレス レジスタをインクリメントする．文字列が等しいかどうかは，DBNE 命令の実行結果でわかる．DBNE 命令で反復処理が終了する条件は文字列が一致しなかった場合もしくは設定された比較回数が終了した場合である．文字列が一致しなかった場合には，データ レジスタ D0 の内容は −1 よりも大きくなる．設定された比較回数を終了した場合は文字列が等しかったことを示す．この場合にはデータ レジスタ D0 の内容は −1 で終了する．

19.4 ASCII から 2 進数への変換

本節では 16 進数を表す ASCII (0～9, A～F) 4 文字を 16 ビットの 2 進数へ変換するサブルーチン プログラムを示す．変換すべき ASCII 文字列は，図 19.3 に示すようにデータ レジスタ D1 に上位桁から順に 8 ビットずつセットされているものとする．変換結果はデータ レジスタ D2 の下位ワードにセットされる．

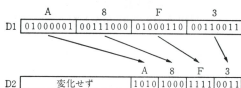



図 19.3 ASCII から 2 進数への変換

変換のアルゴリズムを次に示す．まず $4F で第 6 ビットと下位 4 ビットの抜出しを行う．ASCII 文字が "A"～"F" か "0"～"9" かは第 6 ビット（上位から 2 ビット目）で判定できる．すなわち，第 6 ビットが 1 のとき "A"～"F" であり，0 のとき "0"～"9" である，さらに，"A"～"F" の場合，抜き出したパターンから $37 を算術的に減算したものが $A～$F になり，"0"～"9" の場合，下位 4 ビットがそのまま数値を表す．変換の順序は上位文字から始め，結果を格納する D2 を 4 ビットずつが左シフトしていく．以下にプログラム例を示す．この例では，データ レジスタがバイト，ワード，ロング ワードのいずれでも扱えることを利用して，D1 をデータの保持と演算の両方に用いている．なお，D1 の内容は，変換終了後破壊されている．

```
ASCHEX:  CLR.W    D2          D2をクリア
         BSR.S    AHCV        最上位文字を変換
         BSR.S    AHCV        2番目の文字を変換
         BSR.S    AHCV        3番目の文字を変換
         BSR.S    AHCV        最下位の文字を変換
         RTS                  結果がD2に入って終了
*
AHCV:    ROL.L    #8,D1       D1の最上位バイトを最下位に移動
         ANDI.B   #$4F,D1     下位4ビットと第6ビットを抜出し
         CMPI.B   #$10,D1     第6ビットが1か?
         BLT.S    AH1         下位4ビットが0~9ならAH1へ
         SUBI.B   #$37,D1     A~Fなら10~15に変換
AH1:     LSL.W    #4,D2       D2を左に4ビットシフト
         ADD.B    D1,D2       D2シフトに結果の4ビットを加える
```

RTS　　　　サブルーチン終了

19.5 パリティの生成

本節ではビット データのパリティを生成するプログラム例を示す．ビット データのパリティは，それぞれのビット値の排他的論理和により生成する．すなわち，パリティは，ビット データ内の“1”の総数が偶数個の場合は0に，奇数個の場合は1になる．

ここでは，データ レジスタ D0 の下位16ビットにビット データが格納されているものとし，データ レジスタ D1 をパリティ フラグとして用いる．

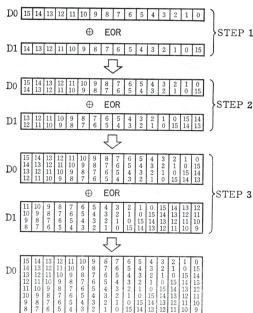



図 19.4 パリティの生成法 (STEP3まで)

パリティ生成のアルゴリズムを図9.4を用いて簡単に説明する．D0にはビット データ d0～15が格納されている．まず，この内容を1ビット左にローテートしたものをD1に格納する．そして，D0とD1の内容をEORし，その結果をD0に格納する．D0に格納された値は，それぞれのビット データの2ビット分のEORとなっている．次に，D0の内容を2ビット左にローテートしたものをD1に格納し，D0とD1の内容をEORしてから，その結果をD0に格納する．ここでD0に格納された値は，それぞれのビット データの4ビット分のEORとなっている．さらに，D0の内容を4ビット左にローテー

トしたものをD1に格納し，D0とD1の内容をEORしてから，その結果をD0に格納する．このとき，D0に格納された値は，それぞれのビットデータの8ビット分のEORとなっている．以下同様にして，16ビット分のEORを求める．データレジスタD0の各ビットは，それぞれすべてのビットデータをEORした結果であり，そのため全ビット同じ値となる．このアルゴリズムに従ったプログラム例を次に示す．

```
MOVE    D0,D1     D0→D1
ROL.W   #1,D1     D1を左に1ビットローテート
EOR     D0,D1     D1⊕D0→D1
MOVE    D1,D0     D1→D0
ROL.W   #2,D0     D0を左に2ビットローテート
EOR     D0,D1     D1⊕D0→D1
MOVE    D1,D0     D1→D0
ROL.W   #4,D0     D0を左に4ビットローテート
EOR     D0,D1     D1⊕D0→D1
MOVE    D1,D0     D1→D0
ROL.W   #8,D0     D0を左に8ビットローテート
EOR     D0,D1     D1⊕D0→D1
```

Ⅲ　68000のサポート システム

20 68000マイクロコンピュータ システム

マイクロコンピュータを利用するには，MPU，メモリ，クロック発生器などを個別に購入し，利用者が自らこれらをボード上に組み合せて，マイクロコンピュータ システムを作ることもある．しかしより手軽にマイクロコンピュータを利用しようとする場合には，LSI メーカやシステム メーカが，広範囲な一般ユーザの利用を目的として発売しているマイクロコンピュータ ボードあるいはマイクロコンピュータ システムを用いると便利である．

この章では，以上のような目的の68000マイクロコンピュータ システムの概要を紹介する．詳細はメーカ発行のマニュアル類を参照されたい．

20.1 H 680 SBC

本節では，日立製作所から発売されている16ビット シングルボード コンピュータ システム（H 680 SBC）について概要を述べる．図20.1にH 680 SB 01 の外観写真を，図20.2にH 680 SBC のシステム構成図を示す．H 680 SBC システムの主要な機能モジュールとして，現在，次の4個が用意されている．(1) シングルボードのコンピュータ本体（H 680 SB 01)：MPU，ROM，RAM，PIA，ACIA，PTM（Programmable Timer Module）などを搭載．(2) モニタ ボード

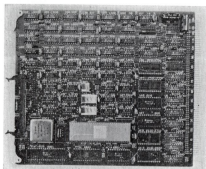



図 20.1 H 680 SB 01 の外観

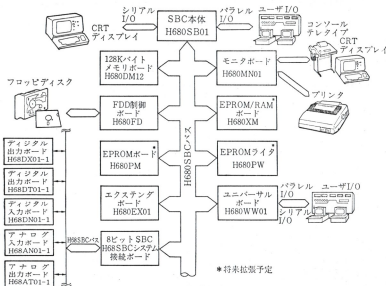



図 20.2 68000 マイクロコンピュータ システム (H 680 SBC システム)

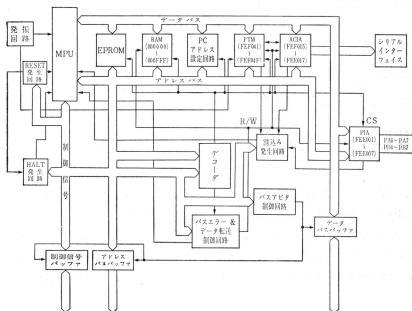



図 20.3 H 680 SB 01 のブロック図

(H680MN01)：標準OSとしてデバッガを実装し，RMS (Real time Monitor System) あるいはテキストエディタ，アセンブラなどが搭載できる．(3) RAMボード (H680DM12)：128Kバイトのダイナミックメモリを実装している．(4) フロッピディスク制御ボード (H680FD)：フロッピディスクの制御を行う．これらのモジュールはコモンバス形式のインターフェイスをもっているため，ビルディングブロック形式で拡張することができる．

次に，H680SBCシステムの核となるコンピュータ本体H680SB01について説明する．図20.3にH680SB01のブロック図を示す．H680SB01内のメモリはEPROM (最大16Kバイト)，RAM (4Kバイト) を合せて最大20Kバイトのメモリを実装することが可能である．アドレスマップを図20.4に示す．システム用とユーザ用に別のエリアが割り当てられている．

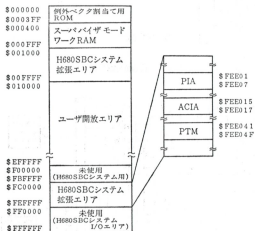



図 20.4　H680SB01アドレスマップ

パラレルI/Oインターフェイス用には，6821PIAが1個実装され，16データラインと3制御線が使用可能である．シリアルI/Oインターフェイス用には，6850ACIAが1個実装され，RS-232Cレベルインターフェイスをもつ機器に接続できる．転送レートは300～9600BPS (Bit Per Second) の間をプログラムにより5段階に設定できる．タイマは6840PTMを1個実装しており，2本の16ビットタイマが使用できる．またボードへの外部割込みは7レベルまで入力することができる．

SBCを複数台用いてマルチプロセッサシステムを構成する場合，ボード上のジャンパコネクタをマルチモードに設定し，モニタボード (H680MN01) を用意すればよい．このときモニタボードに実装されているバスアービタによりバス権の制御が行われる．

20.2 H 680 TR 01

H 680 TR 01 は，68000 を導入する場合の評価・学習用として開発されたワンボードコンピュータシステムである．図 20.5 にボードの外観写真を，図 20.6 にシステムブロック図を示す．以下に機能の概要を述べる．

図 20.5 16 ビットトレーニングモジュール (H 680 TR 01) の外観

H 680 TR 01 は，32 K バイトのダイナミック RAM を標準実装し，さらにメモリ素子の差換えで，128 K バイトまでメモリ容量を拡張できる．また RAM 領域の全空間はシステムとユーザが共用できるモード（モード I）と，スーパーバイザ空間とユーザ空間のそれぞれに分配されるモード（モード II）で使用できる．これらの選択はプログラムでフラグを設定することによって行われる．各モードにおけるアドレスマップは図 20.7 に示すとおりである．システムプログラムは EPROM に書き込まれている．

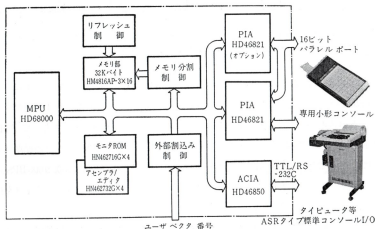



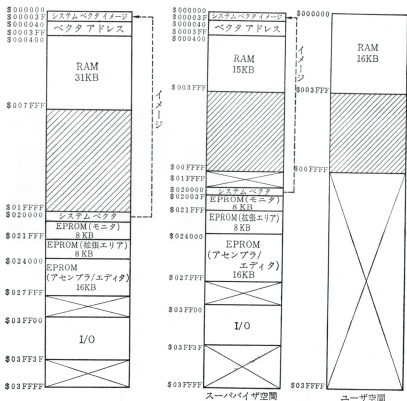
図 20.6 H 680 TR 01 のシステムブロック図

(a) モードⅠにおけるアドレスマップ　　(b) モードⅡにおけるアドレスマップ

図 20.7 H 680 TR 01のアドレスマップ

図20.6に示したように，専用小形コンソールH68PC01をパラレルポートにつないで，68000のマシン語レベルの操作ができる．またオプションのPIAを1個追加すると16ビットバスのパラレル入出力ポートとなる．シリアルインターフェイスとしてTTLまたはRS-232Cレベルのインターフェイスを備えているので，ASRタイプの標準コンソールにつなぐことができる．転送レートは300, 600, 1200, 2400, 4800, 9600 BPSのなかから選択できる．これにより紙テープベースでアセンブラ，テキストエディタを使用できる．

割込み機能としては，ボード内で発生する割込み（システム割込み）以外にユーザが外部から入力する割込み（ユーザ割込み）が使用できる．またパワーオンリセット，リセ

ットスイッチによるリセットおよびメモリの実装されていないアドレス空間をアクセスしたときのバス エラー検出機能を有している.

サポート ソフトウェアとしては，入出力ルーチン，基数変換ルーチンなどのモニタ，アセンブラおよびテキスト エディタが EPROM で用意されている.

なお，16 ピンの TTL の IC を約 30 個実装できるユーザ用のスペースを有しているため，小規模なハードウェアを実装して，システム機能の拡張を図ることができる.

21　68000マイクロコンピュータ開発支援装置

マイクロコンピュータの応用にあたっては，開発するシステムの多様化，高度化に伴ってハードウェアとともにソフトウェアの効率的な開発が一層重要な鍵となる．しかしながらマイクロコンピュータ応用システムの開発では，

（1）ハードウェアが完成しないとソフトウェアのデバッグが難しく，両者の同時開発が困難である，

（2）マイクロコンピュータ応用のユーザ システムでは，通常，ソフトウェア開発のための言語プロセッサやデバッグ機能をもたないため，ユーザ システム上でのデバッグが困難である，

などの問題が伴い，効率的にシステム開発を行うことが難しい．

これらの問題を解決するために，マイクロコンピュータ開発支援装置がLSIメーカ，システム メーカから種々発売されている．これによりプログラムの作成からデバッグ，さらにプロトタイプ システム上でのリアルタイム シミュレーションまで一貫したシステム開発を行うことができる．

本書では，種々の68000用開発支援装置のうち，特に日立製作所より発売されている68000マイクロコンピュータ開発支援システムH680SD300を紹介する．

21.1　H680SD300のハードウェアの概要

21.1.1　システム構成

図21.1はH680SD300の外観写真である．標準入出力機器として，コンソールCRT，プリンタおよびフロッピディスク装置が装備されている．本体部は，15枚のモジュールを収納でき，将来68120IPCモジュールを追加すればコンソールCRT4台，プリンタ

1台がサポート可能となり，マルチステーションが実現できる．

H 680 SD 300 内部モジュールの構成は図 21.2 のように 4 種類 4 枚のモジュールからなり，各モジュール間は標準 VERSA バス† により結合している．表 21.1 に H 680 SD 300 の本体の仕様を示す．

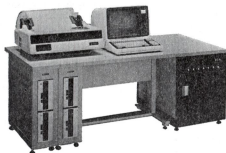



図 21.1　68000 マイクロコンピュータ開発支援システム（H 680 SD 300）の外観図

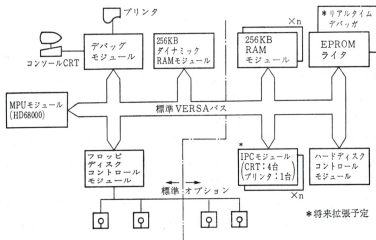



図 21.2　H 680 SD 300 のハードウェア構成図

21.1.2　システムの機能概要

H 680 SD 300 は，標準システムとして，MPU モジュール，デバッグモジュール，フロッピディスクコントロールモジュール，256 K バイトダイナミック RAM モジュールの 4 モジュールを装備している．これらは標準 VERSA バスで結合されるようになっている．

† 標準 VERSA バス：モトローラ社のマイクロコンピュータ開発支援システムで用いるモジュール間のバス方式

表 21.1　H 680 SB 300 の仕様

項　　目	仕　　様
プロセッサ 68000	データ　：1, 4, 8, 16, 32 ビット アドレス　：16 MB（24 ビット） アドレス形式　：14 種類 レジスタ　：17 個+1（ステータス レジスタ）個 命　令　：56 種（1, 2, 3, 4, 5 ワード）
外部割込み	7 レベル（同レベルの場合はスロット番号の小さい方の割込み優先）
バス制御	5 レベル（同レベルの場合はスロット番号の小さい方のバスリクエスト優先）
主記憶部容量	256 K バイト（標準）
入出力ポート	RS-232 C 用ポート×2（標準），ボーレート 50〜19200 ボー
プリンタ インターフェイス	並列ポート（セントロニクス仕様）
電　源	AC 100 V ±10%　50/60 Hz
消費電力	標準構成時（H 680 SD 300 本体，フロッピディスク，CRT ディスプレイ，プリンタ）の合計で 1100 VA
動作温度範囲	10〜35°C
寸　法	高さ 650×幅 1540×奥行 800〔mm〕

a. MPU モジュール　MPU モジュール上には，MPU（68000），クロック発生回路，4 セグメントのメモリ管理ユニット，プライマリ/セカンダリ マップ切換えロジック（次節で説明）およびモジュールの自己診断用のファームウェアがある．メモリ管理ユニットにより，オペレーティング システム（OS）の管理下でのメモリ割当てや，マルチタスキングが可能となる．またリアルタイム マルチタスキング OS によるタスクの同時処理により，プログラムの開発をよりスピードアップさせることができる．たとえばアセンブラでプリンタを使用中に，コンソール CRT でプログラムの編集が行える．

b. デバッグ モジュール　デバッグ モジュールは，デバッグ機能をもつファームウェア，バス管理ロジック，プリンタ用入出力ポート，コンソール CRT 用 RS-232 C ポートおよびホスト コンピュータとの交信用 RS-232 C のポートから構成されている．

c. フロッピディスク コントロール モジュール　フロッピディスク コントロール モジュールには，68000 システムからのデータ要求の処理と，フロッピディスク用の自己診断のために，専用マイクロコンピュータが使用されている．このようなマルチプロセッシング技術を使用することにより，システムのパフォーマンスを向上させ，より効果的なシステムの利用を図っている．

d. 256 K バイト ダイナミック メモリ モジュール　メモリには 64 K ビットのダイナミック RAM チップを使用し，65536 ワード（16 ビット/ワード）のブロック 2 個から構成されている．ダイナミック RAM のリフレッシュのためにモジュール内にリフレッシュ機能をもっている．8 ビットごとに 1 ビットのパリティ ビットを付加して，パリティ

エラーチェックを行っている．パリティ エラーが発生したときは，MPUに対してバスエラーを伝える．メモリ アドレスの割当てはボード上のスイッチで任意に設定可能である．またこのメモリ モジュールは，増設可能で，1Mバイト以上の常駐メモリを使用できる．

e．H680SD300周辺装置　H680SD300で使用する周辺装置の仕様の概要を表21.2に示す．

21.1.3 システムの拡張性

a．マルチステーション　H680SD300は同時に複数のユーザがプログラム開発を行えるマルチステーションに拡張可能である．IPCモジュール1枚の追加で4台のコンソールCRTと1台のプリンタを接続できる．この場合メモリを追加することが必要である．

表 21.2　H680SD300周辺装置

入出力機器名	機能仕様
コンソールCRT	◦インテリジェント機能付 ◦1920文字/画面 ◦14インチ画面，緑色文字
プリンタ	◦両方向印字形シリアル プリンタ ◦ドット インパクト方式 ◦180文字/sec ◦132文字/行
フロッピディスクドライブ装置	◦両画面単密度使用（日立製 FDD 402） ◦500Kバイト/台 ◦IBM 3740フォーマット
高速ラインプリンタ（オプション，将来）	◦700行/min ◦132文字/min ◦活字インパクト方式

b．リアルタイムデバッグ装置　68000ASE（Adaptive System Evaluator）モジュールおよびASE BOXを追加することにより，リアルタイムで68000ユーザ システムをデバッグすることができる．ASE BOX内の68000は，ユーザ システムのMPUとして動作し，コンソールCRT上のキーにより簡単にデバッグできる．

c．EPROMライタ　EPROMライタ モジュールを追加することにより，EPROMに対し，書込み，読出し，照合，ブランク チェック，コピーができる．

d．ハードディスク装置　ハードディスク装置を外部記憶装置として接続可能である．フロッピディスク装置に比べ，高速処理ができ，多くのプログラムおよび大規模なプログラムの開発が容易となる．

21.2 ソフトウェア構成

図21.3にH680SD300のソフトウェアの構成を示す．FDOS, EMS（Exective Monitor System），CRTエディタ，マクロアセンブラ，リンケージ エディタ，FORTRANコンパイラ，PASCALコンパイラ，スーパPL/Hコンパイラが提供されており，さらにシ

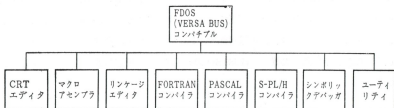



図 21.3　H 680 SD 300 のソフトウェア構成図

ンボリック デバッガなどのオプション機能の開発が進められている．これら各ソフトウェアの内容については，22章で述べる．

H 680 SD 300 を使用して，FDOS，EMS のもとでユーザ プログラムをデバッグする際，プライマリ メモリマップとセカンダリ メモリマップの概念が必要となる．ここでは図 21.4 に示す H 680 SD 300 のメモリ マップを用いて上記各概念について説明する．

プライマリ メモリ領域

$000000
FDOS

・FDOSコマンド
・CRTエディタ
・マクロアセンブラ
・スーパPL/H
・FORTRAN
・PASCAL
・リンケージ エディタ
・シンボリック デバッガ
・ASE
・ユーザ プログラム

RAM領域

$FE0000
システムI/O
$FE8800
EMS
$FEC800
システムI/O
$FFFFFF

セカンダリ メモリ 領域

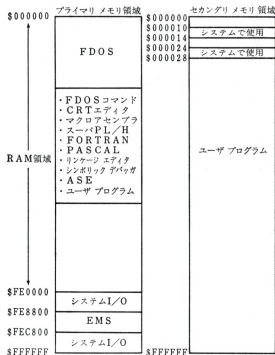
$000000
$000010
システムで使用
$000014
$000024
システムで使用
$000028
ユーザ プログラム
$FFFFFF

注1）H 680 SD 300 上のメモリ ボードのスイッチを切り換えることにより，メモリを 64 K バイト単位にプライマリ メモリ領域またはセカンダリメ モリ領域に割り当てることができる．
注2）RAM は 256 KB 単位に増設可能である．

図 21.4　H 680 SD 300 メモリ マップ

a. プライマリ メモリマップ　プライマリ メモリマップの場合，ユーザ プログラムは FDOS と同一領域にロードされ，ユーザ モードとして実行させる．したがってデバッグ対象のプログラムは，ユーザ モードでなければならない．この場合，FDOS のデバッグ コマンドを使用して，ユーザ プログラムをデバッグする．

b. セカンダリ メモリマップ　セカンダリ メモリマップの場合，FDOS の「SMLOAD」コマンドにより，フロッピディスク上のユーザ プログラムがセカンダリ メモリマップ上にロードされる．この場合，ユーザ プログラムはスーパバイザ モードおよびユーザ モードの両方を使用することができる．さらに 16 M バイトの全メモリ空間をユーザが使用することが可能である．ただし $(000010)_{16}$ 番地および $(000024)_{16}$ 番地からの各 4 バイトはシステムで使用し，ユーザは使用できない．セカンダリ メモリマップの場合，EMS のデバッグ コマンドを使用してユーザ プログラムをデバッグできる．

21.3　H 680 SD 300 の使い方

21.3.1　入出力機器の指定

H 680 SD 300 は入出力機器としてフロッピディスク，コンソール CRT およびプリンタを備えている．これらの入出力機器の指定方法は本システムを使いこなす上で重要な点である．本システムの FDOS では，入出力機器の指定を次のように行う．

a. デバイスの指定方法

（1）フロッピディスク

#FD×× 　（××はドライブ装置の番号 00, 01, 02 および 03）

（2）CRT ディスプレイ

#CN×× 　（××：00 および 01）

（3）ログオン ターミナル

#

b. ファイルの指定方法　FDOS におけるファイル名の標準フォーマットは次のとおりである．

〈ボリューム名〉：〈ユーザ番号〉，〈カタログ名〉，〈ファイル名〉，〈拡張名〉，〈保護キー〉

（1）ボリューム名；フロッピディスクにつけた固有名（1～4 桁の英数字）

（2）ユーザ番号　；ユーザにつけられた固有番号（1～9999 の 10 進数）

（3） カタログ名　；ファイルを管理するための名前（1~8桁の英数字）

（4） ファイル名　；最も基本となるファイル名（1~8桁の英数字）

（5） 拡張名　　；ファイルの種類を示す記号（2桁の英字）

システムで使用している拡張名は次のとおりである.

LO: ロード モジュール ファイル
RO: リロケータブル オブジェクト ファイル
PA: PASCAL ソース ファイル
FT: FORTRAN ソース ファイル
AL: アセンブラ ソース ファイル
SY: FDOS システム ファイル
CF: コマンド ファイル
LS: 言語プロセッサのリスト出力ファイル
PC: PASCAL 中間語ファイル
LL: リンケージ エディタのリスト出力ファイル

（6） 保護キー　；ファイルの書込みに対する保護キー（2桁の数字）

21.3.2 システムの動作モード

H680SD300のソフトウェアには次の2つのモードがあり，それぞれ異なった方法で起動される.

（1） FDOS モード

（2） EMS モード

FDOS モードでは FDOS システム ディスクが必要である. H680SD300では図21.3に示すようなソフトウェアが5枚のシステム ディスクに格納されている. 必要に応じて，各ディスクを使用することができる. EMS モードでは FDOS システム ディスクは不要である. 上記2つのモードの起動方法，停止方法については，本書では省略する. メーカから出版されているマニュアルを参考にされたい.

21.3.3 ソフトウェアの開発手順

H680SD300を使用したシステム開発手順を図21.5に示す.

a. ソフトウェアの作成　ソフトウェアの作成は大きく次の3ステップで行われる.

（1） ソース プログラムファイルの作成

（2） アセンブラおよびコンパイラの実行　アセンブラ，PASCAL, FORTRAN あるいはスーパ PL/H コンパイラによりオブジェクト モジュールを作成する.

（3） ロード モジュールの作成　リンケージエディタを使用して，アセンブラやコ

ンパイラが出力したオブジェクトモジュールを絶対番地形式のロードモジュールに変換する.

b. デバッグの方法 H 680 SD 300には次の3つのデバッグ機能がある.

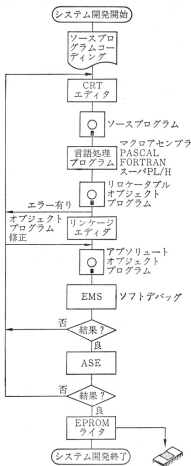
(1) EMSによるソフトウェアデバッグ機能 EMSにはユーザプログラムのロード, メモリ内容の表示/変更およびユーザプログラムの実行等のデバッグ機能がある. EMSの下では, ユーザプログラムはプライマリ メモリマップまたはセカンダリメモリマップ上の絶対番地のアドレスにロードされる.

図 21.5 H 680 SD 300によるシスム開発手順

(2) シンボリックデバッガによるソフトウェアデバッグ機能 シンボリックデバッガはFDOS下で稼動するデバッガで, ユーザプログラムのロード, メモリ内容の表示/変更およびユーザプログラムの実行等のデバッグ機能がある. シンボリックデバッガの下ではユーザプログラムはプライマリ メモリマップ(後にセカンダリ メモリマップもサポートする予定である) 上の論理アドレスにロードされる. また将来はPASCAL等の高級言語のシンボルによるデバッグ機能も可能となる.

(3) ASEによるソフトウェア/ハードウェア デバッグ機能 ASEにはH 680 SD 300に接続したユーザ システムのハードウェアおよびソフトウェアの両方を同時にデバッグする機能がある.

その特徴は次のとおりである.

i) ユーザ システムのクロックで動作し, リアルタイムでソフトウェア, ハードウェア両方のデバッグが可能である.

ii）　2箇所のブレークポイントをハードウェアで設定できるので，リアルタイムブレークが可能である．

iii）　125 ns ごとの高速トレース，MPU インターフェイスチェック，サイクルリセットによる解析機能，タイムアウトチェック機能など多くのハードウェアチェック機能をもっている．

22 サポート ソフトウェア

アプリケーションプログラムの効率良い開発を行うためには，サポート ソフトウェアが充実していることが重要である．一例として，図22.1に日立製作所の68000用サポート ソフトウェアを示す．レジデント システムとしては，シングルボード コンピュータ（SBC）用と開発支援システム H680SD300用のものがある．またクロスシステムとして，大形計算機 HITAC M シリーズおよび IBM370用のものがある．モトローラ社のサポート ソフトウェア システムもほぼ同様である．サポート ソフトウェアの詳細は，メーカから出されている説明書を参考にされたい．本書では，レジデントのなかから，いくつかのソフトウェアについての概要を紹介しておく．

図 22.1 68000用サポートソフトウェア

22.1 FDOS (Floppy Disk Operating System)

FDOS は，大容量で，操作性のよい両面フロッピディスクを記憶媒体としたタスク管理機能，ジョブ管理機能およびファイル管理機能を有し，スーパ PL/H, FORTRAN, PASCAL などの言語プロセッサ，ユーティリティならびにユーザプログラムを統一して管理する．またハードディスクベースでマルチプログラミングをサポートする機能も具備している．その特長として，

（1） コマンドインタプリタ方式の採用により，会話形式でオペレーションができる．また，よく用いる定型的なコマンド群を登録することにより，操作性を向上させることができる．

（2） プログラミングからデバッグまで，一貫してフロッピディスクベースでできる．最大4台(2Mバイト)までのフロッピディスク(両面または片面単密度)をサポートする．

（3） CRT エディタ，マクロアセンブラ，スーパ PL/H, FORTRAN, PASCAL リンケージエディタ，ASE, EPROM ライタおよび FDOS ユーティリティなどのプログラムを簡単に呼び出せる．

表 22.1 FDOS 制御コマンド

項番	分類	コマンド	機能
1	セション管理	(BREAK キー)	セションを開始する
2		LOGOF	セションを終了する
3		BYE	セションを終了する
4	ジョブ管理	CHAIN	カタログドプロシジャの実行
5		RETRY	アボートした現コマンドを再実行させる
6		PROCeed	アボートした次のコマンドから続行させる
7		END	カタログドプロシジャによる実行を終了させる
8		BATCH	バックグラウンドでのカタログドプロシジャの実行
9		CANCEL	バッチジョブの中止
10		QUERY	バッチジョブの状況表示
11		ARGuments	CHAIN および BATCH コマンド用パラメータを作成する
12		NOARGuments	CHAIN および BATCH コマンド用パラメータをリセットする
13	プログラム実行	LOAD	プログラムをメモリにロードする
14		START	プログラムを実行させる
15		CONTinue	中断したプログラムを続行させる
16		STOP	プログラムの実行を中断させる
17		TERMimate	プログラムの実行を止める
18		BSTOP	ブレークキーにより，プログラムを中断させる
19		BTERM	ブレークキーにより，プログラムを止める
20	ディフォルト管理	USEID	当該セションのディフォルトボリューム ID, ユーザ番号の変更
21		DEFaults	当該セションのディフォルトボリューム ID, ユーザ番号の表示
22	時刻管理	DATE	現在日時を表示し，日付を変更する（ユーザ番号0のみ）
23		TIME	現在日時を表示し，時刻を変更する（ユーザ番号0のみ）

表 22.2 FDOSユーティリティコマンド

項番	分類	コマンド	機能
1	ファイル管理	RENAME	ファイル名の変更
2		DEL	ファイルの削除
3		DIR	ディレクトリの内容表示
4		FREE	ディスク空エリア数の表示
5	イニシエーション	INIT	ディスク システム テーブルの初期化
6	ユーティリティ	BACKUP	ディスクコピー/ベリファイ
7		COPY	ファイルのコピー/ベリファイ
8		CONV	M/IBM フォーマットのファイルを SD 300 用に変更する
9		DUMP	ディスク/ファイル内容の表示
10		LIST	ソース ファイルの表示
11		MBLM	S タイプ オブジェクトをバイナリ ファイルに変換する
12	デバッグ機能	PATCH	ロード モジュールの修正
13		DEBUG	FDOS デバッグ コマンドへ制御を渡す
14		SMLOAD	セカンダリ メモリマップ上にオブジェクトをロードする
15	プログラミングシステム	E	CRT エディタへ制御を渡す
16		ASM	マクロアセンブラへ制御を渡す
17		PLH	スーパ PL/H へ制御を渡す
18		FORTRAN	FORTRAN へ制御を渡す
19		PASCAL	PASCAL フェーズ 1 へ制御を渡す
20		PASCAL 2	PASCAL フェーズ 2 へ制御を渡す
21		LINK	リンケージエディタへ制御を渡す

表 22.3 DEBUGサブコマンド

項番	分類	コマンド	機能
1	ブレークポイント	BR	ブレークポイント番地の指定，表示
2		NOBR	ブレークポイント番地の解除
3	トレースおよび条件ブレーク	OP	オプション指定 （1） トレース （2） 特定アドレスのメモリ内容の一致/変化によるブレーク （3） 指定命令数実行後のブレーク
4		VA	上記ブレーク条件 (2) の比較値指定，表示
5		VM	VA で指定した値のマスクビット パターンの指定，表示
6		VL	上記ブレーク条件 (2) のメモリアドレスの指定，表示
7		MC	上記ブレーク条件 (3) の命令数指定，表示
8		XM	例外処理時のブレークマスク指定，表示
9	プログラム実行	G	現在の PC 番地からプログラムを実行
10	メモリ，レジスタ内容の表示変更	MD	指定番地からのメモリ内容を指定個数表示
11		MM	指定番地のメモリ内容を変更
12		DF	全レジスタ内容表示
13		Ri	レジスタ内容の表示，変更
14	プリンタ出力	AP	プリンタへ接続
15		MOP	指定番地からのメモリ内容を指定個数プリンタへ出力
16		DP	プリンタ切離し
17	その他	EXIT	FDOS へ制御を移行（ただしタスクは実行）
18		QUIT	全てを終了し FDOS へ制御を移行
19		ST	タスクのステータスを表示

（4） セクタ単位の効率的なスペース管理を行う．

（5） ファイルをユーザ別に保護・管理する．

（6） ユーザ プログラムを簡単に呼び出せる．

（7） マルチタスク処理を行い，スループットの向上を図っている．

（8） MMU (Memory Management Unit) を使用して，タスク対応に独立した論理空間を与え，メモリの保護を実現している．

表22.1，表22.2，表22.3にFDOSの制御コマンド，FDOS下のファイルを管理するユーティリティ コマンド，ユーザ プログラムのデバッグに用いるDEBUG サブコマンドのそれぞれを示す．

22.2 EMS (Executive Monitor System)

EMSは物理空間用のモニタで，プライマリ マップおよびセカンダリ マップ上でのプログラムのロードやデバッグを行う．またFDOSのブートストラップやプリンタへの出力機能などをもっている．

EMSの主な機能は次のようなものである．

（1） 紙テープ上のオブジェクト プログラムのロード，メモリ内容のパンチおよび照合

（2） アドレス ストップの指定，解除および表示

（3） ブレーク ポイントの指定，解除および表示

（4） プログラムの実行，トレース

（5） メモリ内容の表示，変更

（6） レジスタ内容の表示，変更

（7） プリンタへのハードコピー出力

（8） プライマリ マップおよびセカンダリ マップの切換え

（9） メモリのテスト

（10） FDOSのブートストラップ

22.3 CRT エディタ

CRT エディタは，オペレーティング システムの管理下で動作し，ソース プログラムの作成や変更をCRT画面上で行う．このCRT エディタはコマンド モードおよびページモ

ードの2つのモードをもち，専用CRT コンソールのカーソル機能，制御用キャラクタおよびファンクション キーを用いて，ソース ファイル内の文字や行の挿入，変更および削除を行う．CRT エディタの機器構成を図22.2に示す．機能の概要は次のとおりである．

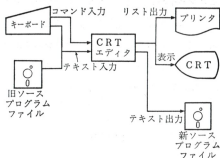
図 22.2 CRT エディタの機器構成

（1） ページ モード

（i） 文字の変更，削除，（ii） 行の挿入，削除，（iii） 1行スクロール アップ/スクロールダウン

（iv） 1ページ スクロール アップ/スクロール ダウン

（2） コマンド モード

（i） 文字列のサーチ，（ii） 文字列の変更

22.3 リンケージ エディタ

リンケージ エディタは，別々にコンパイルおよびアセンブルされた複数のリロケータブル オブジェクト モジュールを結合し，1つのロード モジュールまたは1つのリロケータブル オブジェクト モジュールをつくる．このときセグメントの属性の決定，アドレス空間の計算，ライブラリの検索，外部参照の解決，オブジェクト モジュールの再配置およびエラーメッセージの出力を行う．さらにモジュール マップ，外部定義シンボルのテーブル，未定義または2重定義シンボルなどの情報もプリント出力する．

図 22.3 リンケージ エディタの入出力機器構成

リンケージ エディタの基本的な処理は，次の2つである．

a. 外部参照関係の解決　別

別にコンパイルまたはアセンブルしたプログラム群の間で，別のプログラム中のシンボルを参照（これを外部参照とよぶ）することを可能とする．

b. モジュールの結合　別々にコンパイルまたはアセンブルしたプログラム群を，1本のロードモジュールに編集，結合する．

図22.3にリンケージエディタの入出力関連図を示す．

22.5 マクロアセンブラ

H680SD300のFDOSの管理下で動作するマクロアセンブラの機器構成を図22.4に示す．68000アセンブリ言語で記述されたソースプログラムをフロッピディスクより入力して，機械語に翻訳し，絶対配置（アブソリュート）または再配置可能（リロケータブル）なオブジェクトモジュールに編集して出力する．

主な特長は次のとおりである．

（1） マクロ機能により，コーディングが簡略化される．

（2） 条件付アセンブリングにより1つのプログラムを多目的に利用できる．

（3） リロケータブルなオブジェクトの作成が可能なため，1つのプログラムをモジュール単位に分けてアセンブルし，リンケージエディタにより結合および絶対番地の割当てをすることができる．

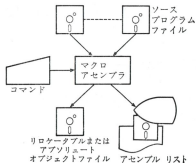



図 22.4 マクロアセンブラの機器構成

（4） リストおよびオブジェクトはアセンブラ制御命令によって制御できる．

（5） 標準登録シンボル数；2000個

（6） 標準アセンブルステップ数：8000ステップ

マクロアセンブラのコマンド，制御命令の詳細は，メーカから出版されているマニュアル類を参照されたい．

22.6 FORTRAN

68000用のFORTRANは，FORTRAN77のサブセットに，ビット処理機能を追加し

た高級言語である．このコンパイラは，ソースプログラムをリロケータブルなオブジェクトモジュールに変換する．データのタイプは，整数形（2バイト，4バイト），実数形（4バイト），論理形および文字形がある．それぞれ変数または配列として使用できる．一方メモリ内のデータをファイルとして扱える内部ファイル機能があり，印字するデータの編集が容易になっている．また構造化プログラミングがしやすい機能や3角関数，双曲線関数で代表される多くの組込み関数等が備えられている．さらにビット処理を行うサブルーチンもある．

22.7 PASCAL

PASCALは，N. Wirthによって設計された高級言語である．系統的にプログラムを設計したり，アルゴリズムを記述したりするのに適している．その特徴は，データタイプとして配列，レコード，集合，ファイル，ポインタ形をもつなど，論理的によく整理されたものとなっている点である．また条件文や繰返し文の種類にみられるように，洗練された制御構造は，構造化プログラミングに適したものとなっている．

68000用のPASCALは，いわゆる標準PASCALとよばれる仕様（UCSD仕様）に，68000アーキテクチャの特長を十分発揮できる特有な機能を追加している．

言語機能の概要は次のとおりである．

（1） 13種類の形定義が使用できる： 符号付整数形4バイト（ロングワード），符号付整数形2バイト（ワード），符号付整数形1バイト（バイト），部分範囲形，スカラ形，配列形，レコード形，集合形，ファイル形，文字列形，ポインタ形，文字形，論理形

（2） 変数は，上記一般形を宣言でき，またプログラマ自身が自由に構成することも可能である．

（3） 演算には，NOT演算記号，乗法演算記号（*, DIV, MOD, AND），加法演算記号（+, −, OR），関係演算記号（=, <>, <, >, <=, >=, IN）が使用できる．

（4） 代入文，手続き文，GOTO文，空文，EXIT文，複合文，WITH文，IF文，CASE文，繰返し文（WHILE文，REPEAT文，FOR文），入出力文等の実行文が使用できる．

（5） 手続きには，関数形手続きと，サブルーチン形手続きとがあり，各手続きは，再帰的呼出し可能（RECURSIVE）で，FORWARD属性を使用することで，手続きをコンパイル単位の外に拡張することもできる．

（6） プログラム中で宣言するすべての変数，手続き，ラベルは，その有効範囲をもつことができる．

PASCAL の概略を次のプログラム例を用いて説明する．

```
Program Call (input, output);                ①
  var i, m, n: integer;                      ②
  procedure cube;                            ③
  begin n: =m*m*m end;                       ④
begin for, i: =1 to 10 do                    ⑤
  begin read (m); cube; writeln (n); end;    ⑥
end.                                         ⑦
```

① プログラム頭部；識別名 Program から始まり，プログラム名 Call，プログラムパラメータ名 input，output と続く．input，output は基準ファイルとよばれる．

② 変数宣言部；ローカルに変数を定義する．この例では，変数名 i，m，n に対して整数形を定義している．

③ 手続き宣言部の頭部；手続き名と引数（オプション）を指定する．プログラム名 Call の実行文 ⑤ 以前に使用する手続きがすべて宣言されていなければならない．

④ 手続き cube の実行文部；複合文の形（⑤ で説明）をとり，m^3 の値を n に代入している．

⑤ プログラム名 Call の実行文部；begin で始まり ⑦ の end で閉じている．この形を複合文とよび，囲まれた各実行文が，1つのまとまりをもつ．第1実行文の for 文は次の実行文を 10 回繰り返す役割をする．この例では，該当実行文が複合文の形をしているので，全体が 10 回繰り返される．

⑥ read (m) は，基準ファイル input から，変数 m に整数を読み込む．読み込んだ値を手続き cube を引用して 3 乗値を n に求める．求めた n を writeln (n) によって基準ファイル output へ出力する．

⑦ 最も外側の複合文（第1レベル）を終了するときには end にピリオドを付記する．

22.8 スーパ PL/H

スーパ PL/H は汎用高級言語 PL/I からマイクロコンピュータの応用システム記述に必要な機能を抜粋し，さらにマイクロコンピュータ特有の機能を付加して構成した言語である．また 8 ビットマイクロコンピュータ 6800 用の PL/H との上位互換，インテル社

の16ビット マイクロコンピュータ8086用のPL/M86との互換性も考慮されている．さらにフロータブル機能†，3次元配列，3レベル構造体なども可能である．

PL/Hは，アセンブラ言語に比べプログラムが読みやすく，少ないステップ数で記述できるため，プログラムの生産性と品質の向上を期待できる．

ソースプログラムファイルから入力したスーパPL/Hのプログラムは文とよばれるプログラムを記述する要素の集まりで構成される．これらの文はスーパPL/Hコンパイラによって，記憶数を確保したり，プログラムの機能を実現するための機械語の列に翻訳され，オブジェクト モジュール ファイルに格納される．

スーパPL/H言語の機能概要は次のとおりである．

（1） 7種類のデータ形を使用できる　符号なし整数形1バイト（バイト），符号なし整数形2バイト（ワード），符号付整数形2バイト，符号付整数形4バイト，実数形，ビット形，ポインタ形

（2） 変数はスカラ変数，3次元までの配列，3レベルまでの構造体として宣言できる．

（3） 演算は算術演算（＋，－，＊，/），比較演算（<>，＝，>＝，<＝，>，<），論理演算（NOT, AND, OR, XOR），番地参照演算（@）が可能である．

（4） 代入文，CALL文，RETURN文，GOTO文，IF文，単純DO文，反復DO文，DO-WHILE文，DO CASE文，機械依存文（HALT，ENABLE，DISABLE，TESTANDSET）等の実行文が使用できる．

（5） 手続きには関数形手続きとサブルーチン手続きとがあり，再入可能属性（REENTRANT）を指定することができる．さらにサブルーチン形手続きには，割込み属性（INTERRUPT）を指定することができる．

（6） ブロックの中で宣言するすべての変数，手続き，ラベル，マクロは有効範囲をもつことができる．

（7） 外部名定義属性（PUBLIC）と外部名参照属性（EXTERNAL）を用いて変数，および手続きの有効範囲をコンパイル単位の外に拡張できる．

（8） 組込み関数，組込み手続，組込み変数を使用できる．

（9） マクロ，INCLUDEライブラリを使用できる．

スーパPL/Hの概略を次のプログラム例を用いて説明する．

---

† ロードモジュール（たとえばROM化したプログラム）をアドレス空間のどこに割り当ててもよいという機能

```
EX1:DO;                                          ①
   /* FOUR RULES OF ARITHMETIC */                ②
   DECLARE(A,B,C,D,E,F)WORD;                     ③
                                                 ④
   A=25;/* A AND B ARE NON-NEGATIVE */           ⑤
   B= 8;/* INTEGER LESS THAN 65536 */            ⑥
   C=A+B;                                        ⑦
   D=A-B;                                        ⑧
   E=A*B;                                        ⑨
   F=A/B;                                        ⑩
END;                                             ⑪
```

① プログラムは EX1: のように〈プログラム名〉:（コロン）で始まり，次に DO; が続く．この DO; と最後の END; の間がプログラムである．1行目のような行を単純 DO 文といい，END; までを一般に単純 DO ブロックとよぶ．

② /* と */ で囲まれた部分を注釈という．コンパイラがプログラムを機械語に翻訳するときは空白として処理する．

③ プログラムの中で使用する6個の変数を宣言している．このような文を宣言文という．宣言文は記憶場所を確保するためのもので，その変数が使用される前に宣言しておく必要がある．A から F までの変数は WORD 形であることを示し，それぞれに2バイト（16 ビット）ずつの記憶領域が割り当てられる．

④ 空白行．プログラムを読みやすくするために挿入してある．

⑤ 代入文．右辺の値を左辺 A に代入する．DO 文や宣言文と同様に，代入文も ;（セミコロン）で終る．この行は代入文の後に注釈をともなっている．

⑥ 変数 B に値8を代入する．

⑦ A の値に B の値を加え，その結果の値を C に代入する．

⑧ A の値から B の値を引き，その結果の値を D に代入する．

⑨ A の値と B の値を掛け，その結果の値を E に代入する．乗算の結果は上位ワードを切り捨てた値となる．

⑩ A の値を B の値で割り，その結果の値を F に代入する．除算の結果は小数点以下を切り捨てた値になる．

⑪ END 文．1行目の DO 文と対応してプログラムの範囲を表す．

付　　録

付録として，258〜261ページに実行命令オペランド一覧表を，262〜266ページにオペレーションコード一覧を掲げる．実効アドレス形式については表 4.1 を参照されたい（45ページ）．なお，オペレーションコード一覧表に用いられる用語の定義は以下のとおりである．

サイズ	00　バイト 01　ワード 10　ロングワード
R/M	(レジスタ/メモリ)：レジスターレジスタ=0，メモリーメモリ=1
dr	(direction)：右シフト=0，左シフト=1
i/r	(カウントソース)：イミディエイトカウンタ=0，レジスタカウンタ=1

項番	命令	意味	ソースオペランド												デ	
			Dn	An	An@	An@+	-An@	An@ d	An@ Ri, d	×××	PC@ d	PC@ Ri, d	#	SR	Dn	An
1	ABCD	10進加算	○												○	
							○									
2	ADD	2進加算	○	○	○	○	○	○	○	○	○	○	○		○	
			○													
	ADDA	アドレスデータ加算	○	○	○	○	○	○	○	○	○	○	○			○
	ADDI	イミディエイトデータ加算											○		○	
	ADDQ	クィック加算											○		○	○(注)
	ADDX	エクステンドビット付加算	○												○	
							○									
3	AND	論理積	○		○	○	○	○	○	○	○	○	○		○	
			○													
	ANDI	イミディエイト データ論理積											○		○	
													○			
4	ASL ASR	算術シフト	○												○	
													○		○	
5	Bcc	条件付分岐														
	BRA	無条件分岐														
6	BCHG	ビット反転	○												○	
													○		○	
	BCLR	ビットクリア	○												○	
													○		○	
	BSET	ビットセット	○												○	
													○		○	
	BTST	ビットテスト	○												○	
													○		○	
7	BSR	サブルーチン分岐														
8	CHK	バウンダリチェック	○		○	○	○	○	○	○	○	○	○		○	
9	CLR	オペランドクリア													○	
10	CMP	オペランド比較	○	○(注)	○	○	○	○	○	○	○	○	○		○	
	CMPA	アドレスデータ比較	○	○	○	○	○	○	○	○	○	○	○			○
	CMPI	イミディエイトデータとの比較											○		○	
	CMPM	メモリーメモリ比較				○										
11	DBcc	減算付条件分岐	○													
12	DIVS	符号付除算	○		○	○	○	○	○	○	○	○	○		○	
	DIVU	符号なし除算	○		○	○	○	○	○	○	○	○	○		○	
13	EOR	排他的論理和	○												○	
	EORI	イミディエイトデータとの 排他的論理和											○		○	
													○			
14	EXG	レジスタ交換	○												○	○
				○												○
15	EXT	符号拡張													○	
16	JMP	無条件ジャンプ														
17	JSR	サブルーチンジャンプ														
18	LEA	実効アドレスロード			○			○	○	○	○	○				○
19	LINK	スタックリンク		○												
20	LSL LSR	論理シフト	○												○	
													○		○	

スティネーションオペランド										データサイズ			コンディションコード					備考
An@	An@+	-An@	An@d	An@Ri,d	xxx	PC@d	PC@Ri,d	#	SR	B	W	L	X	N	Z	V	C	
										○			*	U	*	U	*	
		○																
										○	○	○	*	*	*	*	*	
○	○	○	○	○	○													
											○	○	–	–	–	–	–	
○	○	○	○	○	○					○	○	○	*	*	*	*	*	
○	○	○	○	○	○					○	○	○	*	*	*	*	*	(注)…バイトサイズの場合は不可
										○	○	○	*	*	*	*	*	
		○																
										○	○	○	–	*	*	0	0	
○	○	○	○	○	○													
○	○	○	○	○	○					○	○	○	–	*	*	0	0	SR形式の場合，ワードサイズは特権命令
									○									
										○	○	○	*	*	*	*	*	
○	○	○	○	○	○													
						○	○			○	○		–	–	–	–	–	
						○	○			○	○		–	–	–	–	–	
○	○	○	○	○	○					○		○	–	–	*	–	–	
○	○	○	○	○	○													
○	○	○	○	○	○					○		○	–	–	*	–	–	
○	○	○	○	○	○													
○	○	○	○	○	○					○		○	–	–	*	–	–	
○	○	○	○	○	○													
○	○	○	○	○	○	○	○			○		○	–	–	*	–	–	
○	○	○	○	○	○	○	○											
						○	○			○	○		–	–	–	–	–	
											○		–	*	U	U	U	
○	○	○	○	○	○					○	○	○	–	0	1	0	0	
										○	○	○	–	*	*	*	*	(注)バイトサイズの時，不可
											○	○	–	*	*	*	*	
○	○	○	○	○	○					○	○	○	–	*	*	*	*	
	○									○	○	○	–	*	*	*	*	
						○	○				○		–	–	–	–	–	
											○		–	*	*	*	0	
											○		–	*	*	*	0	
○	○	○	○	○	○					○	○	○	–	*	*	0	0	
○	○	○	○	○	○					○	○	○	–	*	*	0	0	ワードサイズの場合，特権命令
									○									
												○	–	–	–	–	–	
											○	○	–	*	*	0	0	
○			○	○	○	○	○			—	—	—	–	–	–	–	–	
○			○	○	○	○	○			—	—	—	–	–	–	–	–	
												○	–	–	–	–	–	
								○		—	—	—	–	–	–	–	–	
										○	○	○	*	*	*	0	*	
○	○	○	○	○	○													

項番	命令	意味	ソースオペランド												デ	
			Dn	An	An@	An@+	-An@	An@ d	An@ Ri,d	xxx	PC@ d	PC@ Ri,d	#	SR	Dn	An
21	MOVE	転　送	○	○	○	○	○	○	○	○	○	○	○		○	
	〃	CCRへの転送	○		○	○	○	○	○	○	○	○	○			
	〃	SRへの転送	○		○	○	○	○	○	○	○	○	○			
	〃	SRからの転送												○	○	
	〃	USPとの転送					U	S	P							○
				○												
	MOVEA	アドレスデータの転送	○	○	○	○	○	○	○	○	○	○	○			○
	MOVEM	複数レジスタの同時転送			レ	ジ	ス	タ	リ	ス	ト					
					○	○		○	○	○	○	○				
	MOVEP	外部8ビット バスデータ転送	○													
								○							○	
	MOVEQ	クイックデータ転送											○		○	
22	MULS	符号付乗算	○		○	○	○	○	○	○	○	○	○		○	
	MULU	符号なし乗算	○		○	○	○	○	○	○	○	○	○		○	
23	NBCD	10進符号反転													○	
24	NEG	2進符号反転													○	
	NEGX	エクステンドビット符号反転													○	
25	NOP	ノーオペレーション														
26	NOT	論理否定													○	
27	OR	論理和	○		○	○	○	○	○	○	○	○	○		○	
			○													
	ORI	イミディエイトデータとの論理和											○		○	
													○			
28	PEA	実効アドレスのスタック														
29	RESET	外部デバイスのリセット														
30	ROL ROR	ローテート	○												○	
													○		○	
	ROXL ROXR	エクステンドビット付ローテート	○												○	
													○		○	
31	RTE	例外処理からのリターン														
32	RTR	cc回復リターン														
33	RTS	サブルーチンからリターン														
34	SBCD	10進減算	○												○	
							○									
35	Scc	条件付セット	○		○	○	○	○	○	○						
36	STOP	ストップ											○			
	SUB	2進減算	○	○(注)	○	○	○	○	○	○	○	○	○		○	
			○													
	SUBA	アドレスデータ減算	○	○	○	○	○	○	○	○	○	○	○			○
	SUBI	イミディエイトデータ減算											○		○	
	SUBQ	クィック減算											○		○	○(注)
	SUBX	エクステンドビット付減算	○												○	
							○									
38	SWAP	レジスタ半語交換													○	
39	TAS	テストアンドセット													○	
40	TRAP	トラップ														
41	TRAPV	オーバフロートラップ														
42	TST	テスト													○	
43	UNLK	スタックアンリンク														○

スティネーションオペランド										データサイズ			コンディションコード					備考
An@	An@+	-An@	An@d	An@Ri,d	xxx	PC@d	PC@Ri,d	#	SR	B	W	L	X	N	Z	V	C	
○	○	○	○	○	○					○	○	○	−	*	*	0	0	
									○		○(注)		*	*	*	*	*	(注)下位8ビットのみ対象
									○		○		*	*	*	*	*	特権命令
○	○	○	○	○	○						○		−	−	−	−	−	
—	—	U	S	P	—	—	—	—				○	−	−	−	−	−	特権命令
											○	○	−	−	−	−	−	
○		○	○	○	○						○	○	−	−	−	−	−	
レ	ジ	ス	タ	リ	ス	ト	—	—										
			○								○	○	−	−	−	−	−	
												○	−	*	*	0	0	
											○		−	*	*	0	0	
											○		−	*	*	0	0	
○	○	○	○	○	○					○			*	U	*	U	*	
○	○	○	○	○	○					○	○	○	*	*	*	*	*	
○	○	○	○	○	○					○	○	○	*	*	*	*	*	
—	—	—	—	—	—	—	—	—		—	—	—	−	−	−	−	−	
○	○	○	○	○	○					○	○	○	−	*	*	0	0	
○	○	○	○	○	○					○	○	○	−	*	*	0	0	
○	○	○	○	○	○					○	○	○	−	*	*	0	0	
									○	○	○(注)		−	*	*	0	0	(注)ワードサイズのとき特権命令
○			○	○	○	○	○					○	−	−	−	−	−	
—	—	—	—	—	—	—	—	—		—	—	—	−	−	−	−	−	特権命令
○	○	○	○	○	○					○	○	○	−	*	*	0	*	
○	○	○	○	○	○					○	○	○	*	*	*	0	*	
—	—	—	—	—	—	—	—	—		—	—	—	*	*	*	*	*	特権命令
—	—	—	—	—	—	—	—	—		—	—	—	*	*	*	*	*	
—	—	—	—	—	—	—	—	—		—	—	—	−	−	−	−	−	
		○								○			*	U	*	U	*	
—	—	—	—	—	—	—	—	—		○			−	−	−	−	−	
—	—	—	—	—	—	—	—	—		—	—	—	*	*	*	*	*	特権命令
○	○	○	○	○	○					○	○	○	*	*	*	*	*	(注) バイトサイドの場合，不可
											○	○	−	−	−	−	−	
○	○	○	○	○	○					○	○	○	*	*	*	*	*	
○	○	○	○	○	○					○	○	○	*	*	*	*	*	(注)バイトサイズの場合，不可
		○								○	○	○	*	*	*	*	*	
											○		−	*	*	0	0	
○	○	○	○	○	○					○			−	*	*	0	0	
								○		—	—	—	−	−	−	−	−	
—	—	—	—	—	—	—	—	—		—	—	—	−	−	−	−	−	
○	○	○	○	○	○					○	○	○	−	*	*	0	0	
										—	—	—	−	−	−	−	−	

ABCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Destination Register			1	0	0	0	0	R/M	Source Register		

R/M (register/memory): register — register = 0, memory — memory = 1

ADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Register			Op-Mode			Effective Address					

Op-Mode			
B	W	L	
000	001	010	Dn + EA → Dn
100	101	110	EA + Dn → EA
—	011	111	An + EA → An

ADD Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	Size		Effective Address					

ADDQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Data			0	Size		Effective Address					

ADDX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Destination Register			1	Size		0	0	R/M	Source Register		

R/M (register/memory): register — register = 0, memory — memory = 1

AND

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register			Op-Mode			Effective Address					

Op-Mode			
B	W	L	
000	001	010	Dn ∧ EA → Dn
100	101	110	EA ∧ Dn → EA

AND Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	Size		Effective Address					

Bcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	Condition				8 bit Displacement							

BIT操作Dynamic

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register			1	Type		Effective Address					

BIT操作Static

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	Type		Effective Address					

Bit Type Codes: TST = 00, CHG = 01, CLR = 10, SET = 11

BSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8 bit Displacement							

CHK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	Register			1	1	0	Effective Address					

CLR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	Size		Effective Address					

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CMP	1	0	1	1	Register			Op-Mode			Effective Address					

Op-Mode

B	W	L	
000	001	010	Dn−EA
—	011	111	An−EA

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CMP Immediate	0	0	0	0	1	1	0	0	Size		Effective Address					
CMPM	1	0	1	1	Register			1	Size		0	0	1	Register		
DBcc	0	1	0	1	Condition				1	1	0	0	1	Register		
DIVS	1	0	0	0	Register			1	1	1	Effective Address					
DIVU	1	0	0	0	Register			0	1	1	Effective Address					
EOR	1	0	1	1	Register			1	Size		Effective Address					
EOR Immediate	0	0	0	0	1	0	1	0	Size		Effective Address					
EXG Address	1	1	0	0	Address Register			1	0	1	0	0	1	Address Register		
EXG Data	1	1	0	0	Data Register			1	0	1	0	0	0	Data Register		
EXG Memory	1	1	0	0	Data Register			1	1	0	0	0	1	Address Register		
EXT Long	0	1	0	0	1	0	0	0	1	1	0	0	0	Register		
EXT Word	0	1	0	0	1	0	0	0	1	0	0	0	0	Register		
JMP	0	1	0	0	1	1	1	0	1	1	Effective Address					
JSR	0	1	0	0	1	1	1	0	1	0	Effective Address					
LEA	0	1	0	0	Register			1	1	1	Effective Address					

LINK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	Register		

MOVE Byte

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	Destination						Source					
				Register			Mode			Mode			Register		

MOVE from SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	Effective Address					

MOVE from USP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	1	Register		

MOVE Long

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	Destination						Source					
				Register			Mode			Mode			Register		

MOVEM EA to Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	1	Sz	Effective Address					

MOVEM Registers to EA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	1	Sz	Effective Address					

MOVEP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register			Op-Mode			0	0	1	Register		

Op-Mode; Word to Reg = 100, Long to Reg = 101, Word to Mem = 110, Long to Mem = 111

MOVEQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	Register			0	Data							

MOVE to CCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	Effective Address					

MOVE to SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1	Effective Address					

MOVE to USP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	0	Register		

MOVE Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	Destination						Source					
				Register			Mode			Mode			Register		

MULS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register			1	1	1	Effective Address					

MULU

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register			0	1	1	Effective Address					

NBCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	Effective Address					

NEG

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	Size		Effective Address					

NEGX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	Size		Effective Address					

NOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

NOT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	Size		Effective Address					

OR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Register			Op-Mode			Effective Address					

Op-Mode

B	W	L	
000	001	010	Dn ∨ EA → Dn
100	101	110	EA ∨ Dn → EA

OR Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	Size		Effective Address					

PEA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	Effective Address					

RESET

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

RTE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

RTR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

RTS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

SBCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Destination Register			1	0	0	0	0	R/M	Source Register		

R/M (register/memory): register – register = 0, memory – memory = 1

Scc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Condition				1	1	Effective Address					

Shifts, Data Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Count/Register			d	Size		i/r	Type		Register		

Shifts, Memory

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	Type		d	1	1	Effective Address					

Shift Type Codes: AS = 00, LS = 01, ROX = 10, RO = 11
d (direction): Right = 0, Left = 1
i/r (count source): Immediate Count = 0, Register Count = 1

STOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0

SUB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Register			Op-Mode			Effective Address					

Op-Mode			
B	W	L	
000	001	010	Dn−EA → Dn
100	101	110	EA−Dn → EA
−	011	111	An−EA → An

SUB Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	Size		Effective Address					

SUBQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Data			1	Size		Effective Address					

SUBX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Destination Register			1	Size		0	0	R/M	Source Register		

R/M (register/memory): register − register = 0, memory − memory = 1

SWAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	Register		

TAS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	Effective Address					

TRAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	Vector			

TRAPV

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

TST

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	Size		Effective Address					

UNLK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	Register		

命 令 索 引

事項索引

あ 行

か 行

さ 行

た 行

な 行

は 行

ま 行

や 行

ら 行

わ 行

森　亮一監修

マイクロコンピュータシリーズ　〈A 5〉

1		マイクロコンピュータ	
2	P. R. Rony 著 森監訳, 田島 他訳	マイクロコンピュータ教科書 I	定価 2,200 円
3	〃	マイクロコンピュータ教科書 II	定価 3,300 円
4	〃	マイクロコンピュータ教科書 III	定価 3,600 円
5		マイクロコンピュータ教科書 IV	続　刊
6		マイクロコンピュータ教科書 V	
7	森　亮一監修	汎用マイクロプロセッサ	定価 3,200 円
8	森　亮一監修	ワンチップマイクロコンピュータ	定価 3,200 円
9	森　亮一監修	ビットスライスマイクロプロセッサ	定価 3,200 円
10	国分明男著	マイクロコンピュータメモリ	続　刊
11	寺田浩詔監訳	Z-80 マイクロコンピュータ	定価 3,300 円
12	禿, 喜田 田辺, 藤岡 著	16 ビットマイクロプロセッサ	定価 3,300 円
13	D. I. Porat 他著 柏木浩光訳	ディジタル技術入門	定価 4,000 円
14	喜田,萩原,岩崎 著	68000 マイクロコンピュータ	定価 3,400 円
15	田辺皓正編著	8086 マイクロコンピュータ	4 月刊
16	寺田, 禿 宮本, 前田 著	Z 8000 マイクロコンピュータ	近　刊

定価は変更することがありますのでご了承下さい　　　　昭和58年2月現在